

It's More than Just Toys and Food:

Leading Agile Development in an Enterprise-Class Start-Up

Joseph A. Blotner
Senior Manager, Engineering
Sabrix, Inc.
Lake Oswego, OR 97035
(503) 924-4857
joe.blotner@sabrix.com

ABSTRACT

One of the myths of Agile Development is that self-organizing teams do not need direction. The agile development movement focuses primarily on programmers – programmers should do X, Y and Z, and everyone else should do whatever it takes to support the programmers. A fantastic start, since programmers are the people who actually build the organization's product; however, few techniques are offered to the rest of the organization.

The admonishment to managers instructing them to only provide “toys and food” [1] and buffer the team from external distractions implies that leaders in an agile environment should do less work, and be less involved with the team on a day-to-day basis, than in a more traditional environment. In fact a leader in an agile group must do *more* than he/she would in a more traditional environment and must be even *more* involved in the day-to-day activities of the team.

The Sabrix development discipline has strong and deeply involved management as one of its keys to success. Management best practices, when applied appropriately and discerningly, do not limit, but rather enhance, the productivity and job satisfaction of the individual members of the engineering teams. This paper discusses ways a manager can and should help the team be more productive, have a better understanding of their fit in the organization as a whole and develop team members by being active and involved with the team and the rest of the company.

1. INTRODUCTION

1.1 What we already know

The prevalent cookbooks for methodologies such as XP and Scrum only speak to a manager's role as it relates to his/her activities with the Programmers, including coaching (usually on process, rather than implementation choices or design), and buffering the team from the rest of the organization. These tasks

are important, but there is more to coaching than helping with estimates and tweaking the process, and in order to be an effective buffer for both the team and the company as a whole, he/she must know the product in some detail.

Even though XP, Scrum and other methodologies dedicate part of their descriptions to the role of the manager, he/she is really relegated to a veritable guest in the process. Most of the statements about managers/coaches/trackers describe him/her as stepping in to solve problems or clear roadblocks. The main theme is usually a variation of “get out of the way of the team”:

“The measure of a coach is how few technical decisions he or she makes.” [1]

“But perhaps the most important job for the coach is the acquisition of toys and food.” [1]

“...the XP manager must be comfortable stepping in, making decisions...and seeing the consequences through to the end.” [1]

“Scrum runs itself. Management keeps stoking and prioritizing the backlog queue and removing impediments. The teams do the best they can to complete this work.” [2]

“Management: Led by the product manager, it defines initial content and timing of the release, then manages their evolution as the project progresses and variables manifest. Management deals with backlog and issues.” [2]

The intention behind this approach is well-founded -- free the developer from needless distractions so he/she can do what he/she does best. In fact, the basis for the programmer-centric approach, and advocacy of the near removal of management is understandable -- these methodologies were

developed by engineers who lived through decades of Dilbertian management experiences. Such experiences were replete with foolish decisions and managers who did not listen to their employees – who wouldn't rebel against such a situation?

However, going too far in the opposite direction is not the answer. Management is important -- not weak, inconsistent management that interferes with productivity, but rather strong, involved, consistent management, that enhances not only the product, but also the lives of those who build it.

1.2 The problems with this approach

One of the tenets of Agile Development is that self-organizing teams do not need direction, but in reality, this principle cannot hold up for the long term. I have nothing but the utmost respect for every developer and QA person on my team at Sabrix, but they will be the first to admit that they should not be running the asylum. Why?

First, a good developer who wants to make the best product possible, a characterization that fits all the members of our team, will try to fix every bug and add every enhancement brought to his/her attention. While noble, this approach to workload management has a flaw or two, like never being able to ship, since the product will never be done.

Second, team members need an arbitrator in some situations, such as design approach, workload balancing, etc. This need usually presents itself less often at the beginning of a project, and more often as code freeze milestones and release dates draw near. A manager who has not been intimately involved in the product throughout the project cannot possibly offer educated guidance. He/she can only ask for brief summaries of the problems and ramifications, and cannot help but make a decision on whichever argument seems more compelling. A given decision being the correct one happens mostly by chance, and a bad decision can easily lead to finger pointing (“Hey, our manager told us it was okay.” “Hey, I don't know any details about the product and the developers told me it was a good idea; blame them.”). See what I mean?

Third, a good manager frees him/her team from most meetings that are not directly related to product development. Unfortunately, several such meetings are with individuals who need to know how the product works (e.g., to facilitate technical support or build user training materials). Without intimate knowledge of how the product works, its architecture and its design, the manager cannot contribute anything meaningful, and these discussions will end up *requiring* an engineering team member to participate – precisely what we want to avoid.

Fourth, studies have shown that the most important factor in job satisfaction is good management [β]. And, the way most people describe a good manager is someone who understands and appreciates the employee's contribution and values him/her as a team member. The problem with the hands-off management approach is that there is no way for the manager to truly understand an employee's skills and contribution, no opportunity for mutual respect to develop between manager and engineer. The manager may only see the engineer in terms of KLOC, or XP story points¹, and the engineer will only see an aloof person who has no idea what the engineer does all day.

Finally, on the most mundane level, we should keep in mind that, no matter how much freedom you give an individual, and no matter how objectively you measure productivity, every manager must give every employee a performance review on a regular basis. Without detailed knowledge of the individual's work, attitude and fit with the team, the manager cannot possibly do right by that employee or the company at review time.

No one, not even the best manager in the world, can do a competent job if they are always stepping in and out of context, which is why the manager must be closely involved with the team all the time.

After solidly establishing our development discipline at Sabrix, we embarked on a major re-architecture project on our calculation engine. To allow for healthy exploration and experimentation, I told the team that I was going to step back for about a month and allow them to come up with the architecture for the engine; we agreed on what had to be done at the end of this period.

Over the course of the month, I held weekly stand-up meetings to keep up with status. After five weeks, I asked for more details on the framework. In that meeting, I learned that the project was not as far along as I thought, and that our ultimate deadline was now at risk. We discussed the situation and decided that I would step back into my normal role in order to get the project driving forward again. It actually took an entire cycle for the team to stop resenting my involvement and for us to get back into

¹ When I participated in XP, at MeasureCast, Inc., we weighted stories with points, and measured each person's productivity based on how many points they could complete in an iteration.

the groove that had been so enjoyable and successful for us just one short month ago.

The cause of the problems we encountered was my distance from the team during the exploration process. The team explored, but had lost track of the goals we had set at the beginning. Also, we seemed to have different ideas as to what the word *finished* meant.

This incident, along with several other experiences, helped us recognize that the manager-less team concept is simply not viable at Sabrix. More importantly, we have proven that management best practices, when applied appropriately and discerningly in an agile environment, do not limit, but rather enhance the productivity and job satisfaction of the individual members of the engineering teams.

This paper discusses the management methods we use to help the team be more productive, have a better understanding of their fit in the organization as a whole, and develop team members by being active and involved with the team and the rest of the company. Specifically, I will cover why a manager in an agile environment must...

- Keep an eye on product direction
- Keep engineers from fixing everything they see
- Actually get a product released
- Involve the business side to the correct extent
- Buffer the engineering team from time wasters
- Provide the right environment
- Report progress and problems to the rest of the organization
- Motivate the team members
- Know when his/her job really is just to provide toys and food

... and offer some insight into the ways in which these things can be done in an agile environment.

2 Engineering Management at Sabrix

2.1 The Sabrix Development Discipline Solution -- A (Brief) History

The Sabrix development discipline [4] evolved out of a chaotic start-up company building enterprise-class, mission critical software. The discipline is similar to other agile methodologies in its inherent trust of the individual to apply their best skills to the problems at hand with a minimum of distractions and interruptions. What makes it unique and special is that it neither forces the individual engineer into a very small set of work task options nor requires him/her to figure out the boundaries on him/her own. In addition, it involves the business and support portions of the company with the product, without

requiring a person from each group to be dedicated to the development project.

We have a small, brilliant, brave engineering team whose members can accomplish anything to which they set their minds. We also have business and support organizations that have very definite (though ever changing) ideas as to product direction, feature priority and product delivery/placement. In April, 2001, when I first arrived at Sabrix, a developer would literally be half-way through one emergency when she was pulled off of that task and set on a new emergency – by the same person who had given her the first emergency!! Most of these events occurred within hours of a discussion between our CTO and one of our handful of early adopter customers. The result was a product that did not have the features most important to the widest band of the target customer base, and an engineering team that was sick of being yanked around.

When we decided to discipline ourselves using an agile methodology, the executive team expressed concerns that engineering would become a black hole and that input from outside Engineering would be ignored. To address those concerns and support all parts of the company, we have developed and successfully employed a product development discipline that includes, to varying degrees, all the wonderful things XP and its kin promise to the programmers – empowerment, protection from distractions, clear direction, etc. – as well as those it promises the business organization – priority control, flexibility for change, etc.

Briefly, The Sabrix development discipline employs several of the prevalent agile practices – short cycles, unit testing, pair programming, stand-up meetings, buffering Engineering, the Planning Game – but colors the nature of those practices with traditional project milestones. Sabrix delivers enterprise class software to our Global2000 customer base. Such large, highly sophisticated IT organizations will take software upgrades once per year (twice if we push them), which brings some significant challenges to an agile organization. Our customers need to know well in advance when their requested feature will be in the product. We need to be able to have relatively stable scope, plan our work pretty well in advance and build products that will last for some time.

As restrictive as these constraints may seem, we are still very agile, handling bug fixes, fast-patches and, since we are still a relatively new company, deal-breaker requirements with soon-to-be customers in stride with our longer term projects. Our cycles are short, but do not all have the same character. For example, they take on different quality requirements depending on the phase of the project (as we get

closer to release, quality requirements go up). Likewise, pair programming is voluntary at the beginning of a project, but mandatory in the last couple of cycles.

This discipline has helped us evolve from a chaotic, firefighting group that produced mediocre quality software to a sophisticated, agile, controlled organization, building an elegantly architected, high quality product. One of the primary reasons for this success is strong, involved management every step of the way. The engineering manager at Sabrix has a difficult job, and successful execution is key to overall success.

2.2 Providing the right environment

At Sabrix, we have a corporate culture that supports and rewards exploration, creativity, change (even extreme change in the face of challenging deadlines), communication, focus, dedication and individual growth. We also realize these attributes cannot go unmarshalled if a company is going to produce enterprise class software; that is, software for which a customer is willing to accept a six- or seven-figure price tag.

So, how do we maintain the excitement for the engineering team, keep them from attending endless meetings, writing reams of documentation and getting yanked around from fire to fire, while still making sure that Marketing, Sales and Technical Support get all their concerns addressed and receive the (accurate) information they need? The Manager.

At Sabrix, the engineering manager has a one-sentence job description – get the correct product built, on time, within budget and with enterprise-class quality. To meet that objective, this person is responsible for: keeping the development and QA engineers happy, challenged, directed and productive; communicating the activities and progress of the group to other organizations within the company; and bringing input from the rest of the company back into Engineering. Within Engineering, this job description translates into a great deal of involvement in product design and development, product direction, communication, engineer workload, task estimation, decision making (in both product and technology) and, yes, sometimes getting the hell out of the way. This description may send many emotionally scarred engineers running, but it is important to note that “involvement in” does not have to translate to “impediment to”.

First and foremost, it is the responsibility of the engineering manager to foster and support an environment in which the team can leap off buildings and know that they won't go splat on the sidewalk. Our team members are fearless, and are willing to

march to hell and back for their teammates. This type of bravery does not stick around if it's not appreciated and encouraged. Here is a recent success in our team.

As we were approaching our Candidate Build (a.k.a., code freeze) milestone for a recent project, we ran into a technology compatibility problem between the Oracle RDBMS and the J2EE app server to which it needed to communicate. This problem was driving QA crazy, as a certain critical feature would only work occasionally, and they had a hard time nailing down the cause. Greg, one of the developers, finally figured out the problem and explained it to me.

He told me his idea for a solution, which involved re-writing a good chunk of the feature, and using a technology combination which he had heard of, but had never really implemented himself. I poked a couple of holes in his idea, which he figured out how to solve, and then we stared at each other for about a minute considering whether to go forward with his idea (well, I was considering; Greg was writing the code in his head). We had two weeks until code freeze, and I knew very well that the product could not ship as-is. Then, Greg said, “I can do it in time and make it perfect if I have Jeff to help me, but I know he's busy on other things.”

Now, that's a guy really putting himself out there. He is balancing the fate of the product (and, in this case, the company) on the notion that this out-of-left-field idea of his would work. I knew both Greg's and Jeff's work and work habits very well, and understood how the change would affect the product architecture, so I agreed that they should go for it – it seemed the right solution both technically and architecturally, and I believed in them. A manager who does not have an intimate understanding of the work done day-to-day may have balked out of fear of failure, and instructed Greg to look for a different, safer solution. And, after enough such encounters, Greg would stop making those kinds of suggestions, and the company would suffer for it.

By the way, Greg was right. I offloaded the appropriate amount of Jeff's work to other developers, the two of them solved the problem and the feature was bug free by the time we hit our code freeze milestone.

2.3 Keeping engineers from fixing everything they see

Your best engineers – the ones you go to when you have huge problems and no time in which to solve

them – are the people who care most about product quality and customer satisfaction. They hate the fact that their product has to ship with bug notes and believe it is never good enough. Unfortunately, this awesome trait has a downside – lack of ability to prioritize. Every bug he/she sees, and every time he/she hears Customer Support say, “I was on the phone with a customer and they would love it if...” he/she is on the track of taking care of the problem.

XP-ers may argue that the Customer is supposed to provide priority direction, but he/she cannot provide technical direction. In addition, the Customer is usually someone who’s job it is to get as much into the product as possible, so reining in the engineering team from making more fixes and adding more features is naturally antithetical to his/her natural tendencies. It is the manager who must help the developer prioritize. Sometimes, a developer must be instructed *not* to fix something. For example:

With huge companies running hundreds of thousands of transactions through our system per day, performance is a huge deal. During a recent project, we made significant performance improvements. James, one of our developers, took on the responsibility of getting the product to a point where the question of performance would never again be a potential barrier to a sale. After the initial work was done, we saw 8x improvement in throughput. James then found several additional ways to improve performance and wanted to get going on them right away.

He and I discussed his ideas, and, because I know the inner workings of the product in detail, I was able to credibly argue that these additional changes were minor improvements compared to what we had done so far. James knew that as well, and we agreed together that we had done an awesome job on performance so far, and these additional changes could wait until a future release, since we had lots more non-performance work to do on the current release.

The Engineer wants to (and should want to) go all the way, and needs someone to help him/her see where the priorities are, and neither the XP Customer nor the traditional product manager has the technical expertise to understand the true importance of a given bug fix. A good engineering manager has enough technical and business knowledge to weigh each issue against the rest.

2.4 Getting the Right Product Built and Released

Two things all agile methods agree on: 1) you do not want Business making technical decisions or Engineering making business decisions, and 2) there must still be influence in both directions. No matter your development process, every software product must have a plan, whether it be managed by a formal Product Requirements Document, XP’s Planning Game or a simple spreadsheet. Finally, the product must be released in order to be sold.

The problem is, no one explains how to accomplish these dichotomous goals with agility when you can’t have the XP Customer as a full-time member of the team. Scrum’s stand-up meetings are somewhat helpful here, but they are intended to be short and thus should not contain a lot of background discussion to get a non-Engineer (e.g., the product manager) up to speed on an issue.

As a small start-up, Sabrix does not have the luxury of having a customer, or even a customer representative, 100% dedicated to our efforts. Therefore, we also charge the manager with making most of the day-to-day (usually minor, sometimes major) product direction decisions. A good engineering manager in an agile environment can act as the customer when none is available, provided that strong product management is also in place to support the manager when needed.

I can hear the XPer and Scrummers scoffing and gnashing teeth, but the fact is that, in a start-up environment, where everyone in the company is working ten to twelve hour days, there simply are not enough resources to have a dedicated customer embedded in Engineering. In fact, I submit that, with strong product management and a strong engineering manager, a dedicated customer is not necessary in any agile environment.

At Sabrix, the product manager is the strategic customer, while the engineering manager is the tactical customer. What this means is the product manager provides the overall vision and feature-specific requirements, and the engineering manager makes the sub-decisions, as it were, to make sure that vision is realized.

As we were re-writing our web app user interface, we had a decision to make with regard to maintaining page parameters as the user navigates around the system. What we determined is that maintaining two certain pieces of session data were mutually exclusive. I considered the two options, using my knowledge of which option would be most beneficial to our users. I made the decision and the developer moved forward.

In this case, I was able to make the decision on my own, given my knowledge of how our users use the system. I made the decision and the developer was able to move on, without needing to wait for someone else to find time to answer our question.

Just as important as being able to make these decisions, is the manager's ability to recognize his/her inability to do so in certain cases. If strategic input is required to make a tactical decision, the product manager must be brought into the conversation. For example :

The exemption certificate management feature of our application had some usability problems. The developer, QA person and I sat down together to try to hash them out. We came up with a couple of technical solutions, but couldn't decide which to use. In other words, we had some tactical solutions drafted, but needed strategic involvement. So, we called in the product manager and asked her about the future of this feature. She outlined its roadmap for us, and it became clear that there was a major overhaul coming soon. So, we chose the solution that gave us the greatest improvement for the least effort, thereby improving the current customer experience somewhat, while focusing more resources on other areas.

This problem looked small at first, but as we drilled down into it, I realized that it was inappropriate for me to make a decision on direction, because I did not have enough facts. I brought in someone who did have the facts, and we made the decision together.

The product manager and I also discuss issues one-on-one as the need for extra communication arises, and I provide a weekly "newness review" to her, demonstrating the changes in the product over the past week – an XP practice usually left to someone outside Engineering. By keeping in touch with Product Management on a regular basis, the engineering manager has a consistent understanding of customer needs, enabling him/her to better handle the day-to-day product direction decisions, and the product manager can still influence product direction on a continual basis without dedicating 100% of him/her time to doing so.

Within Engineering, I spend the majority of my day listening to design and solution ideas, or answering questions about how best to solve a problem. This feature of our discipline is particularly effective and self-perpetuating. By fielding the questions, I am very aware of the problems being faced by the team. By understanding the problems from the entire team, I have a solid understanding of the product and can

see how different problems may be related. By understanding the product, I am better able to answer the questions and "connect the dots". By staying intimate with the nature, design and state of the product, I can step in and program when time is short². And, by being so intimately involved with the code, the engineering team knows that I can provide well-founded answers and input to their questions, so they bring me more questions. And, finally, the Customer Support team knows I do all these things, so they are confident of getting real answers from me, thus leaving the developers and QA-ers alone.

2.5 Buffering the Team

As with other agile environs, one of the tasks for our managers is to buffer Engineering from the business side of the company. As the engineering manager, I am the liaison between Engineering and the rest of the company – I am the first stop for bugs found in the field and/or information needed to answer customer questions.

When we first established this regimen, end-around attempts were made quite frequently, often by higher-ups in the company. There was one particular instance in which a customer was having trouble with a feature.

The Customer Relationship Manager (CRM) sent mail to the developer who he knew had written that part of the code. What he did not know, was that the developer was out of the office that day.

Later that afternoon, the CRM emailed me asking for the developer's home phone number so he could get an answer to his problem. I went over to his office and asked what the problem was. He told me, and, since I had seen it before, I shared the work-around with him.

I then "gently" reminded him that he is to come to me with his questions. His response was, "I don't care about your processes; I have to satisfy the customer as fast as possible." "So, how'd that work out for you today?" was my response. When he calmed down, I explained that not only does the process prevent wasting time in Engineering, but it also actually gets problems solved faster.

The key to this interaction being successful (if not all that pleasant) was my constant involvement in the

² In fact, I insist on taking at least one development task for each major portion of the application, thereby keeping me even more in tune with the guts of the design

product's development. By being involved, I am able to essentially cache information, and provide quick answers.

The buffer we have set up has been successful in accomplishing its objectives, and it has been accepted by the entire company. That is not to say, however, that members of the business and engineering teams do not mix. In fact, congeniality between the groups has drastically improved since we have instituted this buffer. I even heard one of our developers complement Marketing!!!

2.6 Communicating Outward

The support portion of the company has a need to understand the product. If they do not, then Engineering will spend a lot of time answering support questions, which distracts from our primary mission. This outward communication is a standard practice in all agile methodologies. Exploring this facet more closely, one can see that two aspects of this communication are absolutely essential for success: detail and credibility. The Technical Support organization has to understand part of the architecture of the product and some of the control flow, in order to be able to answer customer questions. In addition, the person who delivers that information must be credible, and such credibility is gained by consistent accuracy of the information provided.

To help enable our Tech Support group, I hold a "Brain Dump" (a.k.a., "Tech Transfer") on the current project some time before release. I demo all new features and explain what has changed since the last release. The session lasts about a day, and my prep time is usually a weekend. The point is -- conveying all of this information to the rest of the company requires absolutely *zero hours from any of the engineering Team members*. It is always greeted with much anticipation, and our Support team reports that the information provided is very useful.

In addition, the Sabrix System must work in concert with 3rd party systems, such as Enterprise Resource Planning (ERP) software and reporting tools. The portions of these external systems (and sometimes the systems themselves) are usually written by our customers' IT departments; and IT groups for Global2000 companies, who have 24x7x365 up-time requirements, are very wary of frequent change. Therefore, changes in our data model and interfacing XML structures must be grouped together, and our customers must know about these changes well in advance.

Again, there must be someone who can relate this information accurately and with sufficient detail.

The various manager-like roles as defined in the currently predominant methodologies (Scrum's project and engineering managers, as well as XP's Coach, Tracker and Customer) must combine to execute this function. However, a strong engineering manager will have an understanding of customer need and technical detail, and can perform this task herself. Having a single individual perform this function is particularly key in a small organization like Sabrix, where we cannot afford to hire three people to do this function.

2.7 Motivating the Team

Different people are motivated by different things. One of my developers is motivated by the challenge of learning something new, another is motivated by the work skill and quality of his peer group, another is motivated by being left alone, while another is motivated by free root beer. For each of these individuals, a different motivational strategy is needed.

Most management books will tell you to be sure to compliment or thank each person on the team at least once per week. Even if they didn't do anything particularly stellar this week, find something small and let him/her know that you appreciated it. Most engineers I've met would see through such a gambit in a minute. Compliments, congratulations, thanks, and even reprimands, must come from a place of knowledge. If I am going to congratulate Tom on re-architecting the central cache, I had sure better know what he did, why he did it and the extent of the effort. And more importantly, Tom must know that I know.

How can that mutual understanding and respect exist? How can I know how cool and elegant the design of the cache framework is, and how can Tom believe that I truly understand it, if I am not in the trenches with him? The answer is -- we can't. The relationship between manager and team member must be "intimate" when it comes to work product and work habits. There is no way for this type of relationship to thrive if the manager is only occasionally or tangentially involved in the day-to-day work of the team.

2.8 Knowing when it really is just toys and food

So..... sometimes, it is most appropriate for a manager to step out of the way and make sure there are plenty of pistachios in the jar on his/her desk for people. A manager should be sensitive enough to see when he/she can be most valuable by just getting in the way of progress.

On a recent project, when we had one cycle to go before our Functional Implementation Complete (a.k.a., Code Complete) milestone,

and all of the major tasks were mapped out, one of the developers asked me one day (in earshot of several other team members) how I was doing. “So, are you going crazy making sure we hit the date?” I told him that I was doing great. As far as I was concerned, they were a well-oiled freight train, and 75% of my job for the next few weeks was to just make sure there were no cows straying across the tracks.

That sparked a little discussion with the surrounding folks about the subtleties of our process, and how this laid back approach would not have worked early on in the project, but was just the right thing at the moment.

Even with as much noise as this paper makes about consistent manager involvement, it should be noted that a strong manager knows when to step back. But I cannot stress enough that this stepping back must be rare, and only done when strategic benefits can be clearly seen. The paradigm should not be one of stepping in only when things go wrong, but rather stepping back only when it is unquestionably the correct thing to do.

3 Good Help is Hard to Find

Throughout this paper, the terms “good manager” and “strong manager” are frequently used. But, what constitutes a “good” or “strong” manager in an agile environment? There are shelves upon shelves of books on good management practices for all business disciplines, and many of those best practices are very applicable to an agile software development team.

The description in this paper of the activities required of a manager are not for everyone. We all know of situations in which a manager got his/her job by being the best developer or seniority. Unfortunately, most people who find themselves as a manager via one of these methods are most likely not going to succeed. They are either unable or unwilling to do all the things required.

The problem is that a strong manager must be able to perform all the tasks mentioned in this paper. The success of the team, product and even company may rely on how good a manager this person is.

Therefore, it is important to identify the correct person to manage the team. The following section contains some traits and skills to use as benchmarks for identifying the person(s) to lead your team. A good manager is a person who has (at least) the following skills and characteristics:

1. Ability to translate customer requests into system architecture
2. Intuition for what does and does not make sense for the product
3. Significant technical knowledge
4. Sensitivity
5. Courage, humility and a thick skin

3.2 Customer Request Translation

When a customer requests new product behavior, it is important that the request is assessed for impact on the current system – both the architecture and planned development. There must be someone who can take the sentence “We want the system to do XYZ” and provide a response back to product management that describes the impact of XYZ on other related features, as well as the overall architecture of the system. Some features may seem small on the surface, but turn out to be significant once the hood is lifted. In a small company like Sabrix, where we must be responsive to our customers, this impact assessment must be done quickly.

For ultimate accuracy, an engineer should be brought in to analyze each request, but one of the principles of agile development is that Engineers are not to be called into every meeting. If we could batch up our requests into one estimating session per cycle, it may be possible to have this done by the developers, but even so, we would lose valuable productivity time doing estimates.

To accommodate both needs -- responsiveness and productivity, we have the engineering manager do this work. When a request comes to me, I try to flesh out the requirements, considering not only the request itself, but its potential impact on other areas of the system. I then derive a few solution options, and provide an estimate on each. If I cannot determine a solution on my own, I seek out an engineer or two to help. Such meetings are required for about one out of every dozen requests, and take about 10 minutes.

Whoever does this manager job must have an understanding of the domain space (in our case, transaction tax management) and a solid understanding of the system functionality, so he/she can respond quickly and accurately to requests and questions.

3.3 Intuition

Customers, particularly large customers, will often request (i.e., demand) product changes from software vendors, particularly small vendors. A company whose goal is to sell a single product to many customers cannot succumb to the specific requests of only a few of those customers. In order for a company to avoid being led astray by such requests, a

filtering process must take place to make sure that the requested change does not break the design intent of the software. At Sabrix, we experience a significant number of requests for specialized features due to the history of our solution space. Our competition forces tax departments to create workarounds for nonexistent features, so when a customer receives our system, they want the ability to create the same workarounds, even though our software renders these workarounds unnecessary.

Now, everyone knows that a salesperson is not going to say something like, "I understand you want that feature, but we're not going to give it to you because it is outside the design center of our product." (If you have a sales person that will take this stand, give him/her a big fat raise and then clone him/her!) Most customer requests make a lot of sense, but many do not, so there must be some way of discerning between these two types. As stated above, the engineering manager is the perfect position for this filter.

In order to succeed in this responsibility, the engineering manager must have a firm understanding of the mission of the product and its architecture, and be able to recognize where new requests come into conflict with either (or both) of them. For example: We have a feature in the Sabrix System called the "Workbench" where sample invoices are created and calculated so users can verify that the tax configuration data is correct. One of our customers pointed out that some of the options in a few of the fields do not make sense to use together (like using a Ship-From address for a service transaction), and they requested that we disable certain fields or limit the choices in certain lists in such situations. While the account manager thought it no big deal, I pointed out that the whole point of the Workbench is that you can do whatever you want, so limiting flexibility like that completely contradicted the mission of the feature.

3.4 Significant Technical Knowledge

The tenets of successful management -- mutual respect between manager and employee, buffering team members from time wasters, and providing valuable input on technical and product decisions -- require that the manager knows his/her stuff. The manager should have been a developer in the product's technology and have been key in designs of non-trivial software.

It's not required that he/she has been a good programmer, but he/she must have the nature of a good designer. The difference is subtle, but important. A good programmer knows how to code up stuff that works; a good designer, on the other hand, sees the bigger picture, and can build systems.

The successful manager does not have to dwell in the bits full time, but has to live in them, moving between levels. He/she must be able to understand the details, as well as the overall design.

One of the ways I accomplish this goal at Sabrix is to take on some development tasks in each area of the product. For example :

When we built our new UI, I was involved in the discussions on how the framework would be designed. Then, the team built and tested the framework. Finally, I built a few of the UI pages so I could understand well how the framework supported the individual UI pages.

Remember that a manager in this environment has to be the arbiter of differences of opinion on designs or other technical issues, so he/she has to be experienced in solving tough problems elegantly and making these types of decisions.

A good manager is one who can get down into the trenches and code with the team (a la the XP coach), as well as step up to the 10,000 foot level and understand the implications of the implementation. Again, he/she does not have to be the best programmer or architect in the world, just very good at both.

The prevailing agile disciplines and processes are strident in their assertion that the manager should not be involved in coding. I disagree, and my experience has proven me out at Sabrix. Again, you cannot throw just any person with the title "Manager" into such a situation, but if you find the right person and free him/her to get involved, all aspects of development improve. The credibility and respect between manager and team member, the design of the system and the productivity of the team all blossom when the right leader is deeply involved.

3.5 Sensitivity

A manager must be in tune with the ever-changing personal dynamic of a group. Even with collective code ownership, a team is not a collection of interchangeable parts. People have different working styles, different productivity "sweet spots" in the day and different motivational factors. Rather than treat the whole team the same, a good manager will treat each individual differently in order to enhance the productivity and quality of life for each person on the team.

Similarly, bringing a new member into a strong team is a tricky business. Even in the most open of groups, a re-adjustment of each person's role in the group will occur, and these changes are not always pleasant

or comfortable. The manager should be able to successfully predict the impact of a new teammate on each current team member, and work to grease the skids whenever possible.

Finally, people change over time. When moving to a new facility, we gave each person on the team their choice of cubicle set-up. One of our developers, Jeff, decided that he wanted high walls in his cube because he likes to do heads-down focused development. He got such a cube, and I kept his preferences in mind for other facets of the environment. After a while, Jeff had an opportunity to get a cube with a window; however, it was more of an open space, with one side fully exposed to other members of the team. Over time in this space, Jeff has gotten even more involved in architectural issues and participated in more design discussions due to this “exposure”. I have noticed that his level of contribution to the product, which was already huge, became even greater, and his enjoyment of his job improved. Now, having recognized this shift, I work to include Jeff more in discussions he may not necessarily have been involved in before.

3.6 Courage, a thick skin and humility

Hey – you are going to be wrong sometimes. But, as a manager, you have to make decisions and communicate information in a timely manner. The key to continuing to succeed and maintaining credibility is to have the courage to step up and continue to make bold, definitive statements. Some management theories state that the most important thing is to make a decision, even if it is wrong.

I agree with this principle but not the level of importance given it by these schools of thought. Decisive action is important, provided that the decision has a pretty good shot at being correct (see previous sections on knowing the customer and the product); however, quick action is not paramount to long-term managerial success. Rather, most important is the courage to continue to make decisions and buffer the team, even if you have been wrong in the past, and to get better at it over time.

Recently, I characterized a customer request as patently contrary to the design of the system. No room for discussion. I successfully convinced everyone involved in the discussion that I was correct. About a week later, I learned that I was completely wrong and that the software should definitely change to accommodate this request. We fixed the software and I sent a public email acknowledging the mistake. The next day, however, I made a similar statement about a different request and was right.

You’ve gotta jump back on the horse! Too many good decision makers have let themselves get squashed by people who attack their ideas by taking criticism too personally. To succeed as a manager in software development, one has to have a thick skin. When I arrived at Sabrix, one of the most critical facets to the success of an agile development environment -- the buffer for engineering -- got shot down, and I was criticized for being what was perceived as obstinate, preventing our product on the fast track to success. There were several situations in which I was berated for sticking to this policy. Situations like these are pivotal moments in a management career -- how you deal with them determines whether you are destined to be a great manager or a pointy-hair manager.

One more attribute of successful managers is that we understand full well that we are nothing without our teams. I have spent a good part of this paper talking about how I have succeeded as a manager in an agile environment here at Sabrix. Well, that’s the point of the paper -- a good manager who is heavily involved in the details of implementation, etc., is crucial to success. But I cannot stress enough that my successes are made possible, and the effect of my failures mitigated, by an outstanding engineering team of development and QA professionals. I don’t make this statement to bolster the image of the team, but rather because the truth is that, as involved as I am in the building of the product, it is the twelve-member team with whom I partner that get the hard part done.

The description in this section brings up one major question -- what if you (or the manager who works for you) cannot be a player-coach? Even a manager who does not have the skills or desire to perform all of these tasks must at least be most of the way there. He/She can then complement his/her strengths with trusted proxies in the areas in which he/she is weaker.

For example, a manager may be a strong system architect, a good motivator and strong enough to buffer the team from time-wasters, but weak in the area of the customer domain. The manager in this example could partner with Product Management to help with his/her role as the tactical customer. Your best bet is always having a single person who can do all of it well, but it is possible to share the duties.

Most importantly, the manager is ultimately responsible for all aspects of the job.

4 Conclusion

The notion that even world-class development and QA professionals can best deliver what is most needed for the company without strong, involved

leadership is a fallacy. Managers who spend too much time in their employees' faces, challenging their skills and knowledge are obnoxious and not to be tolerated, and those who take a completely hands-off approach at the expense of not know what goes on in the group are simply foolhardy.

A strong, useful manager is one who: works closely with each team member; provides critique not on skills, but on designs; challenges not knowledge, but potential complacency; and, provides an environment not of abandon, but of channeled and focused energy. He/she must be qualified to be the tactical customer, and still know when to bring in a real customer or customer representative. And, finally, he/she must be able to talk with the team and the rest of the organization at their level.

The difference between this type of manager and the current advocated brand of agile manager is the same as the difference between a player-coach and a coach in sports. A coach schedules practices, maps out the drills and draws up the plays, but doesn't participate in the scrimmages or games. This type of coach can be successful, but over time, he/she will lose touch with the subtleties of the game and the nuances what it means to be playing. A player-coach does everything a coach does, but stays in the game. He/she is on the court with the team in pressure situations, so he/she is constantly aware of the impact of a given play, or how different parts of a game have different characteristics.

Such a manager is not only acceptable in an agile environment, he/she is vital in it. Because of its downplay of artifacts and documentation and emphasis on verbal communication and collective understanding, it is absolutely crucial for the agile development manager to be a force with the team. Being such a force is a huge commitment; if you are doing it right, you will work as many, if not more, hours than any of the members of the team, freeing them to do what they do best. The product you build, the company who employs you and the team you employ will all benefit immensely.

5 References

[1] Beck, Kent., *Extreme Programming Explained, Embrace Change*. Addison-Wesley, 2000.

[2] Advanced Development Methods, Inc., *ControlChaos.com*, 2003.

[3] Buckingham, Martin and Coffman, Curt. *First, Break all the Rules*. Simon & Schuster, 1999.

[4] Blotner, Joseph A., *Agile Techniques to Avoid Firefighting at a Start-Up*. ACM Press, 2002.