Managing a Large "Agile" Software Engineering Organization

Paul A. Beavers
Senior Director of Software Engineering, BMC Software
paul_beavers@bmc.com

Abstract

This is the story of my business and personal transformation as our department adopted the Agile methodology. The bumps and bruises along the way forced a shift in management philosophy. Embracing the transformation has enabled significant success within the company. The confidence of our customers and internal organizations in our ability to deliver high quality software has increased dramatically. Software releases are now delivered on time with an improved level of quality. However, this success did not necessarily come easily. There were many obstacles to overcome as this large organization transformed itself from a largely waterfall development organization into a high-output Agile development machine. This article presents this transformation and the impact it had on the organization's leadership and management styles.

1. Introduction

Driven by pressure from external customers and internal sales managers, BMC Software's Distributed Systems Management business unit adopted a strategy to modernize its successful but dated PATROL product line. The engineering team had to deliver rapid incremental improvement. The intense requirements included the need to reduce the effort required by customers to install, configure, and manage their systems management solutions. At the same time, the team knew existing customers must be able to leverage their investment in currently deployed solutions.

Business pressure for rapid improvement was growing. As a result, the organization would be required to demonstrate significant improvement in a very short timeframe. It was clear traditional software development methodologies simply would not work. Under the influence of Israel Gat and other "Agile" thought leaders, the team adopted Scrum as the chosen Agile methodology. Today the product line is considered revitalized. There have been four successful releases and the team is on a release cadence which will deliver three releases per year. This presentation will focus on key management concepts which enabled an organization of over 200 engineers in 4 locations to overcome such a monumental challenge.

2. Indoctrination by Fire

The timeframe was September, 2004. Israel Gat had joined BMC Software as the Vice President of the Distributed Systems Management business unit. During his first months at the company, Israel began to introduce the concept of Agile development. In particular, he focused on Scrum. After about three weeks on board, Israel was asked to address the organization in a forum that was simply to be a meet and greet type of meeting. In front of room full of about fifty engineers, Israel talked about releasable software every two weeks. Personally, I was skeptical and did not understand how this new methodology could apply to a complex set of products such as the ones delivered by BMC Software. Not surprisingly, this was the opinion of many other skeptics throughout the organization. It has often been said by Israel to the organization, "we reserve the right to wake up smarter tomorrow morning". Thankfully, this is the style in which Israel manages his organization. When it comes to Agile development, I am quite proud to say I have exercised that right. The following months presented challenges. I had to somehow reconcile my own experience and belief system against this new methodology that at the time seemed loose and haphazard. Questions like "You mean to tell me I am going to test the entire product suite once every two weeks?" , "Agile development may work for small UI based products but what about complicated heavily integrated product-lines?", and "Certainly, these

AGILE 2007
0-7695-2872-4/07 $25.00

product lines require months of design work in order to ensure all of the pieces function properly?" were commonly heard in the hallways. Many members of the engineering staff were quite concerned. This occurred to such a great degree that some staff members simply gave up during the process and sought opportunities elsewhere. We now call them Agile casualties.

As time progressed, one key fact was becoming very clear. This methodology was for real. We were not going to wake up one day and realize this approach which was challenging our thinking was only a dream. Israel continued to plant the ideas of Agile in all of our minds and nurture them until they started to grow. We had over a dozen engineering projects underway at any given point in time. Most of these were managed using a waterfall approach. There were varying degrees of success with our own methodology. Highly disciplined teams were able to deliver software on time within the specified budgets. However, release overhead was high and the projects still suffered from the challenges most waterfall projects encounter - a large investment before any functionality is visible and late cycle quality / integration issues.

In early 2005, one of the engineering managers decided it was time to move forward with Agile. The product was BMC Performance Manager. It contained a base infrastructure as well as plug-able modules which allow for monitoring of a wide array of information technology objects. Once the manager of the infrastructure group made the decision to move forward with Agile, the leaders of the organizations charged with delivering the plug-able modules believed they must adopt the same methodology in order to ensure proper synergy throughout the organization. At this point in time, we had just made the decision to adopt Agile development for a very large scale project. We were going to use Scrum to build the next generation of PATROL, BMC Software's flagship systems management product line. This decision would involve bringing approximately one hundred software engineers up to speed on the methodology in very short time. This was an organization of high-caliber engineers with many years of experience. Not only was the organization very experienced, they were very much set in their ways of doing things.

Inspiring change for this organization was not an easy undertaking. Could we really pull it off? At the time, one thing was very clear. Failure to deliver measurable functionality in a short time could have devastating impacts on both the success of the product in the market as well as the confidence of internal consumers in the engineering organization's ability to deliver anything. After years of struggling to make commitments and deliver on them, the engineering team simply could not afford another failure. Additional challenges were presented as well. The product proposals and scope had all been written and commitments were made to internal consumers. We were going into this with a very fixed scope and a fixed set of consumer expectations. This reality would haunt the management team throughout the entire first release and beyond. However, the fixed scope using fixed resources within a fixed timeframe was impossible. Waterfall methodologies would not have worked in this situation either.

We were in trouble. We had to be successful...and we had just chosen to adopt Scrum as our Agile development methodology! For me personally, I was skeptical. However, my confidence in the team's ability was strong. If Agile was what we needed to do to be successful then we would implement it. We would simply manage the heck out of it. We would produce progress reports, pay very close attention to defect arrival and closure reports, and measure anything else that we deemed relevant at the time.

3. Harsh Realities

The first year with Agile was difficult. There were approximately ten individual Scrum teams working on the project. Coordination seemed impossible. Many of the teams were allowing a high number of defects to exist in their code at the end of each iteration. The management team was becoming restless. Managers feared the unknown. How much progress were the teams making? How could we tell what progress had been made? Six months into this and we did not know where we were. The primary concern was what percentage of the functionality would be complete when the deadline arrived?

Quality was a rising concern. The defect backlog was growing at an alarming rate. The product builds were very unstable and took a long time to complete. The Quality Assurance team members were very uneasy. They spent significant time waiting for builds. Agile concepts were losing popularity amongst the team. This revolutionary methodology that had been so popular in the beginning seemed to be letting us down. Company expectations for delivery were

growing. At the same time, the engineering team's ability to complete working software was ostensibly shrinking.

Stress was building. Many of the team members were feeling pressure from external consumers. We had put in place an overall Steering Committee for the project. This committee was comprised of senior managers responsible for the series of Scrum teams which made up the overall project. Each week the Steering Committee would meet and make decisions regarding the overall project and process improvement. As one would expect, the weekly Steering Committee meetings were growing tense. Tempers were high. We all felt the pressure to deliver but seemed to be stuck in a management vacuum. Each week we would meet and define action items. With many other work responsibilities, the committee members frequently did not complete the defined action items. The Steering Committee was ineffective.

I, as well as others, visited with Israel frequently. After listening to long monologues regarding everything that was going wrong, Israel would simply say to me "Trust the methodology". After that he would point me to Dean Leffingwell, the consultant we hired to advise us on our Agile implementation, for guidance on what to do next. Today, the three words appear often in my conversations with team members when they approach me with a set of concerns regarding the projects. Quite frequently I give the same advice that was given to me "Trust the methodology". This is typically followed up by a discussion around how we go about solving the particular problem at hand.

4. Transformation

4.1 Applying the Lessons Learned

The first release continued to progress, and struggle. Steering Committee outcomes were becoming less and less valuable. One topic was repeated often. Everyone knew we must have an effective nightly build process. If nothing else, the daily battle over who broke the build had to be eliminated. The need for a successful nightly build process was intuitively obvious. However, as a room full of senior management and high level architects, we simply could not figure out the best way to fix the problems. After hours of spirited debate, we would simply end up acknowledging the need for improvement. One day, in one of the Steering Committee meetings we made the decision to assign some of our top talent to the task of making sure the build process was working properly. The Steering Committee created a small Scrum team charged with improving the build process and associated communication regarding build status. The Scrum team would listen to requirements regarding the build process and implement them. The results were astounding. We were soon able to build the entire product every night.

Today, the build process runs with rarely an issue. Since the application is a web application, team members come to the office each morning, look at a build report, point their browser to the latest URL and begin to test working code that was checked into the source code system the prior day. Automatic regression tests have been run against the code and the teams are ready to test the features that were added the previous day. Having our build process functioning like this made a dramatic difference in the amount of working software that could be delivered within an iteration. Additionally, the overall stress level of the group reduced significantly. We learned two primary lessons with the build process success we had created. First, was the obvious benefit of a nightly build process that allows the engineering teams to test working software every day. Second, and perhaps less obvious, was the manner in which we were able to achieve the nightly build objective. After many months of debate, the Steering Committee had realized its own limitations in determining the right course of action for nightly builds. In essence, we decided to have confidence in the engineering team and empower them for success.

Having spent many years in management and believing the best way to achieve success in software engineering was to be directive and provide technical answers; this was a major change in thinking for me. I had reached a level of management where the best approach was to give the teams the direction and tools they need to be successful and the empowerment to make their own decisions. This was a revelation. We had to trust our technical people to be successful.

Our nightly build process involved so much more than simply producing a set of binary images to be installed. The nightly build process served as the point in time when the work from five or more Scrum teams all came together into a single installable image. This integration effort occurred with the production of every installable image. Integration across so many teams

every night was the primary cause of the build problems we faced. In addition to empowering the team to solve the technical issues, the management team's dedication to this nightly integration process proved to be a critical aspect of improving the ability to test the overall product suite every day. The managers demonstrated commitment and desire to not revert to their own independent compile processes when challenges with the integrated build reached significant levels. With an understanding of the priority and an appropriate sense of urgency, each of the managers who now had become Scrum masters, ensured each of their teams had the right focus on the integrated product suite. Through this process we learned we would save significant time and reduce quality risks if we approached the integration of the product components as an ongoing prerequisite as opposed to some future objective. Nightly integration of the software components continues to be a very high priority for all of our Scrum teams.

With a successful build process in place, the first release made it out the door on time. However, the process was painful. Countless hours of stress and hard labor were used to achieve the delivery. However, the functionality in the product simply did not measure up to the expectations of internal consumers. Varying degrees of success were felt by the team members. Many of them believed the original goals had been met. The issue of managing the consumers' expectations related to the product deliverables would be addressed by focused efforts of the leadership team. It was possible to insulate the engineering team from the non-engineering related pressures as we began preparation for the next major release. Just in time for Christmas.

4.2 A New Challenge

Up until this point, my responsibilities had been limited to a subset of the Scrum teams responsible for the overall product. My primary role was to manage the teams that would build modules which would plug into the overall infrastructure. The infrastructure team was lead by a peer of mine. We all reported to Israel Gat, the Vice President. During the holiday season of 2005, I received a telephone call from Israel. He told me he believed the overall product needed a single engineering leader with singularity of purpose. The total organization was made up of about 250 engineers. At the time Israel believed a single Director in charge of the product line would bring the necessary focus needed to continue our quest with both Agile development and revitalizing our product line. By this time, my belief in Agile as our overall methodology for building software had grown dramatically. Believing I had gained an appreciation for the key principles of Agile and that I could somehow make a difference by applying them, I eagerly accepted the challenge that was placed before me.

4.3 Simplifying the Organization

We had a May deliverable for the second release and it was already the first of January. Given the recent change in management, it was obvious we needed to reorganize. As one might imagine, restructuring an organization of this magnitude was a significant undertaking. At the time, many people were of the opinion we should inflict only a small amount of turmoil on the group.

Over the first release, the teams had struggled with understanding the lines of responsibility for the overall system which was made up of many smaller components. The organizational structure contained remnants from many previous leadership teams. This was a large global organization with engineering teams in Austin Texas, Houston Texas, Pune India, and Tel Aviv Israel. Frequently, teams shared responsibility for delivering the same component. In some cases, shared responsibility translated into no team having responsibility.

We were attempting to execute Agile projects with teams spread across the globe. Simple tasks such as having a daily standup had become complicated. With the project clock ticking for the second release, we knew we had to act quickly if the organizational structure was to be changed. In my view, the organizational change was necessary. In a matter of days we defined and implemented a simple organization which allowed each of our locations in different time zones to function as independent engineering Scrum teams. In the Pune location, the team would be responsible for maintaining and developing the existing legacy product set while the U.S. teams were responsible for driving the next generation of the product. Organizing like this was a risky proposition. In particular, there were concerns regarding employee morale and the normal slowdown in productivity resulting from a change this pervasive.

The decisions were made and we restructured the entire organization. When employees, managers, and

consumers looked at the new organizational chart, it was logical and easy to understand where the responsibilities were to be handled. Additionally, each of the Scrum teams now had a clear direction on what they needed to do to contribute to the overall solution set. At the same time, we were able to insulate team members focused on the new generation of product from many of the day to day distractions associated with a large installed base.

As the second release was ramping up, we had established a simple structure within which to work the Agile methodology. This structure enabled the teams to become empowered and have ownership in what they needed to deliver. The reporting relationships were now logical and understandable with clear lines of communication.

4.4 Requirements Management

When the first release was nearing completion, the teams struggled significantly with requirements acceptance criteria. Debates regarding whether a requirement was complete were frequent. Often times, the debates became heated causing turmoil between the teams. It was quite clear that for many features, we did not have a clear understanding of the function it was to perform. In some cases, there were issues with scalability due to the fact that we did not have an accurate understanding of the target customer environment. While these issues can exists with most methodologies, the impact on Agile was significant due to the fact that there was no longer an extended QA period to resolve such issues at the end of the engineering cycle.

These issues were primarily driven by the fact that there was so much ambiguity in the requirements. We had not made the investment up front to fully elaborate the features into detailed requirements. This situation was so extreme that at one point there was a feature that said "Ensure the product performs all of the functions of previous versions". This would have been satisfactory from a product management requirement perspective. However, this feature was never elaborated into detailed requirements. There was constant debate over both the meaning of the requirement and whether or not the requirement was met. Quality Assurance engineers were frustrated trying to determine whether or not we should accept this feature. Simply put, we had not invested enough of our velocity in either up front or ongoing requirements elaboration.

Going into the second release, we knew we had to do a better job with managing, elaborating, and prioritizing requirements. This had strictly been an issue regarding where we chose to invest our energy. Properly managing requirements was critical.

In response to this shortcoming, we decided to implement a new role within the Scrum teams. Essentially, there would be an individual within each Scrum team that would be the "Requirements Architect". Tasked with owning the elaboration of features into requirements, this new role would bridge the gap between a high level feature description and detailed requirements which could be prioritized and managed via the Agile methodology. The primary intent was to have a relative sizing of features and requirements prior to release planning and detailed just-in-time elaboration of features into detailed requirements before each iteration.

As we progressed through the second major release, we discovered the importance of elaborating features into concise requirements which could be prioritized, estimated, and testable when delivered within the Agile iterations. Additionally, we determined we should create requirements which are small enough that they can be designed, built, and tested within each iteration. Large and ambiguous requirements simply do not lend themselves to the Agile methodology.

Our new approach to managing requirements proved successful. We were able to successfully bridge the gap between marketing or product management feature and the detailed requirements that are required for Agile. Another major benefit to our approach was the opportunity for the engineering team to influence the requirements and related priorities. Establishing the Requirements Architects role helped to improve the working relationships between the product owners and the engineering teams. Most importantly, it was the focused attention to the requirements management process that allowed us to determine the method of requirements management that worked best for our organization.

4.5 Qualitative and Quantitative Metrics

With a simple organizational structure and a solid approach to requirements management in place, we entered into the release planning exercise for the

second major release of our products. We had completed the work required to produce a good set of elaborated requirements. The requirements were prioritized in order of importance. We entered into a release planning process where the teams placed the requirements into iterations.

A number of the requirements were prioritized as "Critical". For us, "Critical" meant we would not ship the release unless the requirement was implemented. While this is approach is somewhat waterfall in nature, the end-user consumers had insisted we meet a minimal credible state before release. This was primarily driven by the fact we had disappointed them with the feature set in the first release.

The results from the release planning were surprising. Basically, the teams presented an iteration plan that indicated we could deliver the critical requirements in the allotted time frame. During the presentation of the release planning results, I polled the team for their confidence level in the high-level plan. On a scale of one to five (fist of five), five being the best, how would you rate your confidence level that we can deliver a high quality release containing the critical functionality before the deadline? To this day, the results of this informal survey surprise me. The average confidence level was less than two and a half. Even though we believed we had built a reasonable release plan, the team lacked confidence in our ability to achieve the release objectives.

As a result, we met with the business owners to inform them they needed to either remove the constraint of minimal credible requirements or change the target date. In negotiating with the business owners, we were able to shift the date by adding two additional iterations to the release. When we executed the release, we completed the "Critical" features and several of the non-critical features before the deadline. Adding the two iterations enabled the team to execute with confidence. By the way, the confidence level after adding the iterations was about four.

We learned significant lessons going through the process of determining the team's confidence level and acting on the results. It was clear to us this qualitative approach of simply asking the team their confidence level was a much better approach than only looking to the reports and resource allocation to determine our confidence level. The team had completed the activity of laying out the release. However, they had not felt empowered enough to say "It cannot be done". Paying close attention to the qualitative assessment proved to be the right approach. The release was successful. Additionally, the teams' confidence in their management's ability to recognize concerns and act upon them increased significantly.

As the release progressed, there would be a number of times where we would perform similar assessments to gauge the team's confidence level in various aspects of the release. The technique has been used many times to determine quality levels, release readiness, and release completeness. While it has not been used to replace all quantitative measurement, the qualitative assessment is equally important when making decisions.

Today, we release software on time on regular intervals. The teams drive themselves to meet the dates. The primary exit criteria is "Does the team believe the product is ready to ship?". In order to get an accurate answer to this question it is important to listen closely to what the teams are saying and create an environment where the team members are comfortable being completely truthful. Additionally, it is important to understand the answers and ask probing questions. Often times the answers lie in what is observed more so than in what is specifically stated.

4.6 Trust Your Intuition

Undertaking an effort of this magnitude, implementing Agile across such a large organization, presents many challenges. By design, Agile does not contain rigid rules on how to accomplish each and every detailed task. Many times throughout the transition we were faced with situations where there was no quick answer. There simply was not a manual or document where we could find easy answers on how to do things. In the beginning, this proved to be very difficult. We would be required to rely on our own interpretation of various situations and do what made the most sense.

Often times when faced with complicated problems, I would seek advice from Israel. Many times Israel would say in response, "Paul, trust your intuition". As simple as this may sound, it is probably the best advice I have ever received as a manager. Over time, I began to develop a new mindset. The mindset is to trust your intuition. Not always, but more frequently than not, if something does not feel right, it isn't. My personal

style has transformed into one where I use my intuition and other information to determine where to focus my attention.

For example, when we were wrapping up the release planning for the second release and the team lacked confidence in our ability to deliver, I had noticed small indicators amongst the group which caused me to request the qualitative analysis. When looking around the room, I noticed the body language of the team members. It was clear even though they were saying they had a reasonable plan, they were not confident. These indicators had created a nagging feeling in me that something simply was not right. At the time, I could not specifically define what it was that was causing my anxiety. I only knew that the anxiety existed. The situation did not feel comfortable. My personal comfort level was low. My intuition told me to dig deeper, to try to understand why the team was so uncomfortable. My intuition told me to perform a qualitative analysis. Once the analysis was performed, it was clear the team felt as if they were over-committed. As a result we adjusted the plans. Had I simply taken everything at face value, the impact of disappointing our consumers a second time could have been devastating.

5. Reflections

After reaching this point in which the organization is truly reaping the rewards of its transition to Agile development, it is clear the transition has dramatically altered my personal management philosophy and style. For many years, I believed the best way to manage a group of people was to closely monitor and control the day to day activities of the group. As management, we would set ourselves outside the team exerting pressure in areas where the metrics told us things were not progressing properly. It is clear today, the better approach is to provide high level direction and allow the teams to determine their course of action and self-monitor their execution.

Today the management team sets themselves within the team as active team members all striving to achieve the common goal of delivering working software. We value input from the team members who are actually doing the work. The management team knows the best source for guidance when making product and process decisions is the people who work with the code and product every day. We know our personal and career success is derived from the success of the teams. We must empower and trust them to make good decisions.

Along the way, we have learned a few basic principles in which we follow as our success continues. While some apply directly to Agile development and some are pointed to general management style, the application of all of them will help to ensure success in Agile software development.

Establish a simple organizational structure to enable Agile development. The reporting relationships should be logical and understandable with direct lines of communication. Limit exceptions to the model which defines the functional structure.

Have confidence in the engineering team and empower them for success. Provide each team with the necessary requirements and prioritization and allow them to manage their backlogs.

Trust your intuition and act quickly. Avoid the desire to acquire too much information before making a decision. Let the risk associated with a wrong choice drive the effort put into making the decision.

Pay close attention to qualitative measurement and let go of traditional quantitative metrics. The intuition of the team members is infinitely valuable when determining quality levels and release status.

Stay involved as an active team member. There is immeasurable value for the information gained when managers stop in to a daily standup meeting, an iteration planning session, or iteration demo and participate as if they were an active team member. It is important to remember to behave as a team member and not direct the process.

Manage and prioritize software requirements in a way which is conducive to Agile software development. Follow the guidelines for writing good software requirements (Independent, Negotiable, Valuable, Estimatable, Small, Testable).

Invest in an automated nightly build and verification process. Avoid under estimating the

importance of having regular successful product builds and the effort required to produce them.

Maintain release readiness. Having a market releasable product at the end of iteration should be the goal. Defect prioritization should focus on fixing the defects that would prevent a product from releasing first. Drive toward full regression testing within the iteration using automated testing techniques.

Trust the methodology. When projects get chaotic, avoid the tendency to revert to old style software development. Allow the Agile process to work to address key issues.

With four successful releases behind us, it is clear the adoption of Agile was the right thing to do. It has had a dramatic impact on the morale of the teams and the overall quality and quantity of our deliverables.

AGILE 2007
0-7695-2872-4/07 $25.00 © 2007 IEEE