

# The Rollercoaster of Required Agile Transition

Rich Jochems & Shane Rodgers

*Progressive Insurance*

[richard\\_b\\_jochems@progressive.com](mailto:richard_b_jochems@progressive.com)

[shane\\_p\\_rodgers@progressive.com](mailto:shane_p_rodgers@progressive.com)

## Abstract

*How do you move from a waterfall world to an agile world rapidly without upsetting the apple cart and all the people that are involved in the project? This Experience Report focuses on the transition of a project team being forced by management to use agile principles and practices and the trials and tribulations of the experience.*

## Project Background.

The project involved converting an insurance policy workflow system, POWR (Progressive's Online Workflow Routing) from a fat client SmallTalk application into a thin client C#.Net application. The project included 8 developers, 2 QA analysts, 1 Business Representative, 1 Business Systems Analyst and 1 Iteration /Project Manager and 3 consultants. Most of the senior tech leads were mainframe developers that had only been on waterfall projects their entire careers. To further complicate matters the POWR team was new to C#.NET development, so not only were they learning a new methodology, but they were also learning C# as they went along. The project was to be completed within 12 months and the business customers would receive an application that offered more value by providing "Customer Self Service" functionality and retaining the same core features as the original application. The value to Progressive was getting features of the application off of the mainframe, providing "customer self service" enhancements that eliminated IT intervention in business operations, eliminating 4000 fat client installations, and moving the application into a web browser. This project also brought the POWR application into compliance with enterprise wide standards.

## The Story.

The project started off using a standard waterfall methodology in the initiation phase. Approximately 2 months into the project, after countless requirements and analysis meetings, the decision was made to use Agile methods and techniques. Progressive had a few

other departments trying Agile practices and now our director decided that it was our turn to test out the "new methodology". IT management and the business were on board with trying Agile methods. All agreed it would be pushed down from the PM and the business, to the project team.

A decision was made by management to explore the hiring of consultants to help coach and mentor the team in Agile practices and principles. The project leadership team discussed various options with other project teams inside of Progressive that had already used Agile consulting services. We made our selection based on the most pressing needs of the team. The leadership team felt that mentoring and development resources would be needed from the consulting group that was hired.

The team was then asked to give up their cubes and move into an Agile Alley. It took over a week for everyone to succumb to the fact that they were now going to co-exist without walls, phones and headphones. They would also have to learn a new methodology and new coding language, all within in an area the size of small conference room, with their fellow co-workers for the duration of the project.

**Lesson Learned # 1** – If the team is resistant to change, gradually introduce new concepts.

Management should have seen the warning signs that the adoption of agile methods with this project team was going to be difficult. Most of the senior tech leads were mainframe developers that had only been on waterfall projects their entire careers. To further complicate matters the POWR team was new to C#.NET development, so not only were they learning a new methodology, but they were also learning C# as they went along. From the start of the project the development team felt they were forced into using unproven agile development methods on a project that needed detailed design and requirements and a thorough understanding of the entire project by both the business and IT.

Due to the complexity and size of the project, the development team felt that the project needed a significant amount of analysis and that storycards would not be sufficient and the project would have no direction. Initially the management team, along with the consultants, tried to bridge the analysis gap by having the entire team try to identify as many high level story cards as possible. The consultants convinced management that our iterations would only be two weeks in length. Therefore, the project team was asked to start breaking work down into manageable chunks that could be done in two weeks. The team then broke it down into smaller pieces from there, keeping in mind that each piece was not to be more than 2 days in length. After this process, the development team still wanted to see the big picture through detailed design and requirements as they had in the past.

**Lesson Learned # 2** – Get more buy-in from the entire project team.

Getting the team to buy into the storycard writing process was extremely difficult from many angles. Turning over the requirements reigns to the business, and the entire team, was a difficult concept for the developers and analysts to grasp. Starting development with only a few lines of detail, and the promise of a “future discussion”, scared the hell out of the developers. QA was also getting involved in the storycard writing process upfront, this brought constant questioning and challenges to the storycards. Initially this caused even more uneasiness to the analysts and developers who thought that they were supposed to be the experts. Therefore the development team now had to view QA as allies used throughout then entire process.

Before the management team decided to use agile methods to execute the development of this project, traditional requirements were created for parts of the project’s functionality. These requirements were initially used to pull out high level stories for the first release. The requirements were also going to be used to help create storycards throughout the release, but after iteration 1 started they quickly became obsolete. When the team started collaborating on storycards and the developers started coding, the storycards started evolving, unlike the documented requirements. After several iterations the original

requirements didn’t even look like they were related to the same project. Therefore the giant three ring binder full of documents was placed in a drawer and never heard from again. The team realized that collaboration and working software was truly more important than comprehensive documentation. This was discovered through constantly changing storycards and having the business address changes as they occurred.

We started to gain buy in once we could prove that the methodology was working by having working software. The development team started seeing real progress and the customer started to work with the product that we were producing. Release 1 was 8 iterations and the Agile practices didn’t start gaining real acceptance until we were at the end the release. The forced approach at the start helped aggravate the development team and pushed off the acceptance of the methodology. Had the management team more closely considered the impact of forcing new development methods and new technology, a more phased approach may have been used. By using a phased approach, we may have slowly gained acceptance for some of the practices, this may have given us more time to introduce the more controversial practices, like paired programming, test driven development and automated testing tools.

Testing had their own resistance points that they were facing, and in a sense they were set-up to fail. Management and the consultants insisted on automated testing, of which QA had no experience in. QA also had to choose one of the internally developed automated testing tools and become proficient with it in several weeks.

**Lesson Learned #3** – Provide the right training for agile testing methods and tools.

The consulting team gave some high level overviews on the benefits of automated testing. The internal team that created the testing tool offered plenty of assistance and training on the automated testing tool. Our testers knew the tool inside and out. So when the project started everything seemed like it was going to be just fine in the QA department.

At the end of the first iteration QA had several small story cards completed, the larger ones were still being tested and were moved into the next iteration. By the end of iteration 2, all the larger storycards from the

first and second iterations were either in testing or not even started yet. The analysts and consultants stepped in to help and start cranking out the storycards the testers became extremely frustrated. After spending time with QA it turns out they were testing every single scenario they could think of, similar to waterfall testing. They were testing beyond what was in the storycard. Only testing what was in the storycard was a hard concept for them to understand at first. After many painful iterations they eventually changed their mindset, but that was because they had to, they had to get caught up.

Testing was also trying to put manually written test plans into the automated testing tool. And why not? That's what they thought they should do. They were trained on how to use the tool, but never trained on why and how automated testing should be leveraged. The tool they were using was not meant to hold extremely detailed test plans that they were accustomed too. The tool was meant for adding basic tests to eventually build a fully automated end-to-end system test.

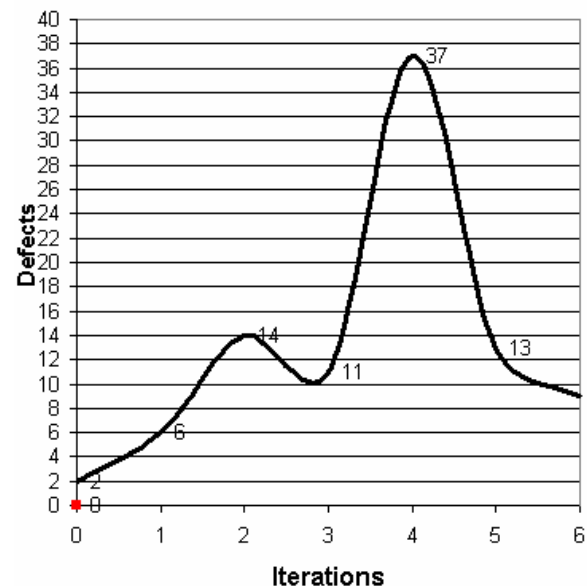
The first three iterations required mandatory pairing pre-assigned at kick-off for the length of the iteration. Due to many complaints and lots of push-back, management thought that mandatory pairing on all story cards was the only option in order to get developers to pair. The team was uncomfortable with pairing, mainly because it was new and they had to collaborate with fellow team members. The team also felt that there was too much micromanagement and they didn't own the process enough. Most of the development team felt that pairing on all cards didn't make sense and that they should be the ones to determine if a story card required pairing. Because of these complaints, management decided to experiment with different process changes to find what works.

**Lesson Learned #4** – Allow the project team to evolve and own the process.

For the first 'experimental' iteration (Iteration 4) we removed most of the processes that people were complaining about. The development team could decide if pairing was needed on a story card or not. If they decided to pair, they could also pick their own pairing partners. From the start of the iteration, basically zero pairing occurred. About half of the stories the developers decided to pair on at the kick-

off, but the pairing only lasted a day or so at the most. Eventually people were wearing headphones and everyone was on their own, back to pre-agile development.

There were several problems we noticed right from the start. The biggest issue was the higher defect count (refer to chart below). This was a noticeable problem because the defects were piling up in QA. This slowed QA progress which also created frustration and tension between QA and the developers that were creating the defects. Another noticeable problem was that the features developed in the "pair at will" iteration were less consistent with the rest of the application. We were not sure if this was due to individuals' style, lack of input from other developers, or just coincidence. One thing was certainly nice; there were no complaints about pairing!



One issue we didn't notice until several iterations down the road was that the features developed during the first 'experimental' iteration all had individual experts that could only work on those features. Eventually others learned the code, but this certainly demonstrated the lack of knowledge transfer during just two weeks.

After the iteration of no pairing we went into the next iteration (Iteration 5) with a slightly different approach. There was now mandatory pairing, but the developers picked their own pairing partner and

they assigned their own story cards. The results were noticed by everyone. The defect count on features created during this iteration went down by more than half the amount. The paired programming also led to better application flow, more knowledge transfer and less frustration in QA. This third method offered the development team more ownership of the process than the first method, yet higher quality products were produced than the second method.

Still another small but useful way to let the team own the process is by turning over stand-ups to the project team. During the first release we ran stand-ups like we were running any other meeting, management lead and got the group to follow. By letting the project team take control of the stand-up meeting, it not only gave them a sense of more ownership it also made for a more effective meeting. By letting different team members take the lead, it helped people understand the benefits of the daily stand-up. With the spotlight on the team rather than management, this seemed to increase the lines of communication and make the meeting more effective. With these types of small changes, the group was slowly maturing into a self-organizing project team.

**Lesson Learned #5** – Select a consulting team that meets the needs of the project team.

Consultants were brought in to help mentor, train and become a part of the project team. Management selected the consulting company based on the fact that they would provide several developers to help train and code alongside our developers. The lead project consultant had a strong personality and pushed more of a “just do it” management approach without understanding the personality of the project team. Just do it may work for Nike, but it definitely didn’t work for a project team that was not accustomed to being forced to do anything.

After compiling retrospective information and evaluating the project overall, we came to the conclusion that management should have based their selection of consultants more on the needs of the project team. By taking a closer look at the personality and background of the project team, they would have realized that the drill sergeant lead consultant would not work. They should have looked for a consulting company that had more of a coaching and mentoring approach versus immediate hands-on

project delivery. With consultants that were more focused on mentoring and coaching the transition to using agile practices may have been less painful. Due to the consultant team that we chose and the project team’s resistance to their approach, the team was slower to evolve to the agile practices and principles than what was originally planned for.

**Lesson Learned #6** – Keep the lines of communication open and quickly address issues.

Open communication is a key principle in an agile environment. Keeping the lines of communication open between all team members is critical to a project’s success. This is especially important when the line of communication becomes strained between the two technical leads, the issue needs addressed immediately. Unfortunately we discovered this the hard way. The two senior tech leads differed in opinion and work ethic from the start and refused to leave their egos at the door. This led to a disruption in the lines of communication and a division in the development team, which eventually led to rework.

We resolved this conflict by having the two technical leads alternate leading stand-ups and by pairing them on critical parts of the application. Seeing the lines of communication open between these two individuals and the success they were having, led the rest of the team to follow.

### **Retrospective.**

As a result of the issues discussed in this paper, it eventually led to the departure of several senior developers. Friction between management and the project team remained even after their departure and certainly didn’t help the project or methodology acceptance curve. It took time for the team to grasp some of the agile principles due to the many issues that occurred during the project. During the life of the project the team’s ability to adapt to the situations helped us reach our goal of delivering a high quality product ahead of schedule and under budget. The team also delivered more functionality to the business than expected.

A large amount of the success was due to the team's ability to openly reflect at retrospectives, which relates to Agile Principle #12:

*At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly [1].*

Through many retrospectives, the team discovered its weaknesses and eventually found solutions. At the start of the project there wasn't a single iteration that went by without multiple adjustments being made somewhere. The team had no problems openly reflecting on progress (or lack of), which helped create an environment that embraced change. The team was learning how to be more effective with each change, whether it was a struggle or not. All parties could also openly discuss and resolve issues at any time. If an issue wasn't resolved in the retrospective, the management team took it as an action item and worked on it until it could be resolved.

Agile practices were eventually accepted, but in retrospect the transition could have been easier if Agile Principle #5 was closely followed from the start:

## References.

[1] <http://www.agilemanifesto.org/principles.html>

*Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done [1].*

At the start, the project team could have had motivated individuals with more buy-in to agile methods and practices. With sufficient training, testing could have avoided the slow acceptance of automated testing. Even though both groups eventually did succeed, management's embrace of this principle earlier on could have avoided the bumpy start.

The project team now understands the value in many of the agile methods. After we were able to adapt agile practices to our team, rather than adapting the team to the practices, we were able to function as a cohesive stable project team. They not only learned new development methods, they also learned a new technology and delivered a high quality product to the business customer.

Today, the team members have "drank the Kool Aid" and have become advocates for agile methods and processes.