Letters From The Edge Of An Agile Transition

Abstract:
Starting as a new coach of a team with almost no agile experience and an enormous release pending was quite daunting. Change was slow and at first it seemed like we would not succeed in converting the team and delivering the product. It took patience, support at many different levels of the organization, and hard work from many different people to put this team on the path to becoming a functional and productive agile team. Now the team is growing as a successful and productive agile team. I am proud to have been a part of going from mostly offshore waterfall development to a local agile development team; from one release in about a year and a half to an average of one release per week; from over a hundred open issues to dozens and falling.

First Baby Steps:
I was brought on by Gale Cengage at the end of May of last year as a developer and coach. The organization was undergoing an agile transformation started by the newly hired CIO. A local consulting firm was brought in to assist with the transformation. I was the second developer and coach brought on directly by the company. I was placed with a team working on a new back end server platform for the company's online products. This platform provides search and retrieval services for educational databases serving the library and university markets, as well as expanding into government and direct consumer markets. The development group was around 40 people, around 30 of whom were developers. They were looking to add an additional 22 people and I was one of the first to be hired.

My first surprise upon starting work with the team was that the team had no idea what was going to happen or what I was there to do. When I arrived it had been just four days since the cube walls had come down. Most of the development team looked shell shocked. Not only were they busy fighting numerous fires in an attempt to get a release out the door, but now the latest new company initiative had descended upon them.

The release itself was over a year in the making and had already been postponed more than once. I arrived with the updated release date only two weeks away. As I began to assess the climate I recall in particular a series of questions I asked that illustrated some of the problems I would face. It started with "Is there a list of what still needs to be done?" "Well we're code complete..." Not quite the answer I was looking for. In an attempt to seek some clarity I rephrased the question. "So what else needs to be done to release to production?" Awkward silence. We were mere weeks from the release and no one person seemed to know what was left to do, yet as I looked around everyone was busy fighting fires.

This lead me to ask about some of the usual suspects: bugs, load testing, and deployment. I learned there was a list of bugs but there was ongoing discussion of which ones were "showstopper" bugs that would hold up the release. Few if any of the bugs had any information on how long they might take to resolve. I was told we wouldn't know until we looked into them. Fair enough, but surely we had an idea of the level of effort involved? "Yes." I moved on to the load tests, which were probably the brightest spot in this line of questioning. Tests were following a plan and ongoing. The catch here was the QA team was waiting for data to be loaded. Loading would take days and may need to be restarted if bugs were found in the loading and indexing process. Not exactly confidence inspiring. Finally I heard the war stories of deployment.

I found I didn't know where to begin. Here I had expected a team ready to start an agile transformation and what I found was a team on the verge of both success and failure: success in the sense that they were "done" with all of the requirements for the release and failure in that once again they were not tracking to deploy the new system by the time the business had agreed upon.

I immediately found the only thing I was sure of was the chances of the release happening in two weeks were slim to nil. The individual developers on the team were
talented, but they were having trouble closing the existing issues without introducing new problems. They didn't have an automated regression suite so they were forced to do manual regression testing, which was both expensive and time consuming. This also had the effect of exacerbating the problem of bug prioritization. The risk of breaking something and having to restart days of data loading was daunting. If something went wrong during data loading, the risk of not making the release date and having to make additional compromises would skyrocket. Without a list of tasks and estimates it was very hard to figure out how the release was tracking. Many people were trying different ways of planning, but since they were not involving the team with the planning process it was not effective.

For example, another newcomer to the team at this time was originally supposed to lead the performance side of the team. He had a lot of management experience and was also shocked by the state of the team in regards to the release. He spent a week or more generating a reasonably complete list of what needed to be done. It was managed by him and done over email. It was better than before but still was geared more to informing upper management of the state of the release than to aiding the team. While he worked hard to get input from the team, the team was not involved in the process, so data could have been more accurate. It was not directly tied to what the developers were doing so while there was a relationship it was only a reflection of what was going on, a snapshot of what was not accomplished.

I was torn; I knew that many of the agile processes could help but it was also clear that this was not a team that was ready and waiting for a switch to agile. Initially I tried to introduce some basic agile practices I thought might help them get a handle on the release. Given the short time frame, scarce unit test coverage, legacy code base, and general resistance to change, it seemed prudent to focus on practices that were not development-specific. This was especially true since the developers were far more concerned at this point with the release than contemplating changing the way they approached their craft.

I embarked on trying to establish some story cards for the work that was currently being done and to get estimates on that work. The hope was that we would be able to get an idea of how we were tracking for the release even if we couldn't immediately put together a specific plan. Also this was relatively unobtrusive change. Developers could more or less continue working as they had been. The story cards would let the team know when a task was finished. The developers agreed to help out but after a short time it was obvious they weren't keeping up. Cards did not have the time the developer had spent on them marked. Most of the cards remained in the "On Deck" section of our story board. Most of the story cards had estimates greater than a day but were claimed to be "done" at a rate of two or three in a day. The obvious problem was estimation, which shouldn't have been a surprise at all. Asking people who haven't estimated before to estimate, especially when they were already under pressure, turned out to be somewhat of a disaster.

The more subtle problem which would only grow in importance over time was the definition of done. The business and development sides of the team were not on the same page as to what "done" was. It was defined as it is on many teams: done was when time was up and you had to release. I wanted to redefine done as a managed and discussed item that was determined before the task started.

While trying to get my head around this new problem of estimation I realized like many of the team's issues it was not one item that was the problem but a conglomerate. In this case many of the outstanding issues being worked on were bugs. Put this together with the weak test coverage mentioned earlier and it would have been hard even for more experienced agilists to have been very accurate with their estimates. Even estimates on more known quantities such as bugs with a known root cause varied greatly. This is due in part to the nature of work with bugs. The bugs had a tendency to go in fits and starts. Without the unit tests or an end to end regression suite, the risk of introducing new defects was high. This had the effect of pushing up estimates in order to account for time spent manually testing changes. Take these problems and add the more subtle issue of the definition of done and you have a very tough nut to crack.
Over the next weeks the pressure mounted and I found it harder and harder to engage the team in discussions of how to keep making small incremental improvements. Frustration set in. I talked with the team lead and the head of development letting them know I may need some support from them. Basically I needed some help getting the herd moving in the right direction. They were understanding but here I had another unpleasant surprise. They wanted to wait until after the release to start changing to agile. This was not a wise decision for the team. No matter how much pain the team went through I knew it would also reinforce bad habits to continue on the path that was already not working. There was a good chance that instead of thinking "wow do we ever need to change" they would think that they had made it through the worst and were still standing. My thought was if we could introduce some change and actually make their lives better along the way to the release we might be able to use this pain as a springboard for change.

Sadly, I was not able to implement agile changes as early as I thought would be most effective. The team lead tried to give me a hand but with his other responsibilities he was not able to give me enough support. Therefore I was forced to pick and choose my battles very carefully as I waited for the release when the team will be declared "agile." This was probably the lowest point in the transformation. The initial release illustrated many of the challenges ahead. The release did not go out the door for another two months - not weeks.

Sparks But No Fire:

As we transitioned from dealing with the release to adding new features and enhancements to the product our problem set also changed. Now that people had a moment to reflect they began to question how this new agile process was helping them. What they did not realize was how many more changes the team had to make before they were truly agile. Here we had just started planting the seeds of change and already people on the team were wondering where the crops were. Previous teams I had worked with were more eager for change and therefore were more likely to be excited with the progress they had made instead of wondering why larger rewards had not appeared. The decision to become agile came from management, not from internal team decisions, and the team was not convinced it would benefit them.

As I considered where to begin, I gained my first ally. Another coach was floating from team to team trying to provide support for the transformation. The two of us had many conversations about the problems the team was having and came up with two leading problems to tackle. One: we just did not have enough people with agile experience to lead by example and/or mentor the non-agile developers. The teams consists of around eight rotating contractors, two full time employees in addition to myself, and another couple of newly hired developers. Out of a team of 13, three had agile experience: me with over ten years, one with a couple years, and one with less than two. The second conclusion we drew was that the team needed a win that would be able to aid the team as it moved forward. This led us to start a project to introduce acceptance and regression testing via FitNesse. This proved to be a challenge as neither of us had the domain knowledge required and we were met with resistance when we attempted to draw in developers. Did I mention the team moved directly from release firefighting to patch purgatory?

Eventually we were able to allocate enough developer time to get the first regression test done. It wasn't much but showed that not only could it be done with agile methods, but that the results were immediately useful.

Given how averse the team was to change, I realized I was fighting an uphill battle to switch the team to agile alone. I had generated some sparks but they were not enough to start the fire. The technical lead of the team was excited about switching to agile; he had experienced enough pain trying to work the old way and saw the potential benefits of a fully agile team. The problem was he wasn't available most of the time. As much as he wanted to change, he also wanted to keep the team moving. Looking back we should have figured out
how to support him better and get some items off his plate since he was already an ally in
the conversion.

The project manager was ready for the change but was also so overloaded that she was
constantly fighting to stay afloat. Even though she saw the benefit of switching she had little
time to spare to help make it happen. In hindsight she was another ally if she had had more
free time. She could have greatly assisted us to help break down/translate
requirements into stories.

Both of these examples reinforced a concept that I knew but had not appreciated. People
who are already behind or overloaded have great difficulty changing their behavior even if
the result would be less work. As a coach you have to find them some time. This meant
picking up work from them at times or figuring out a way to head off some of their work
upstream. In my case I groomed the board, making sure the PM only had to take the cards
I gathered week to week and enter them in a series of spreadsheets and our agile planning
tool to generate reports. Unfortunately due to trouble with the company's choice of agile
tool even this was too time consuming. Tool problems aside it was difficult to find ways to
help these team members without getting drawn too far into their problems. Just taking
over some of their work was not going to keep the team moving forward, but it did help.

Basically between budget and the amount of work that was scheduled for the team the
team needed to more than double its size. This meant adding six to ten people. Previously
these resources had existed in the form of offshore contractors. Now the team was looking
to hire local developers with agile experience. This proved challenging. Agile has become
quite the buzz word and lots of people claim agile experience. We needed more than just
people who had done a little agile; we wanted a good community.

At first the development organization as a whole, not just our team, was able to recruit a
number of people through our friends and family network. After that ceased to provide new
people the organization starting relying on recruiters. The quality of candidates dropped
sharply and the lack of resources started to put projects at risk. Our team was somewhat
insulated given that we had just released (even though we still had plenty of work to do),
but other teams with dependencies on us were ramping up for releases in the near term
that would increase the pressure on us.

The Crucible Of Growing Pains:

One of the problems with integrating new developers was transferring knowledge of the
products from the four team members who had been the primary developers on site for the
last two to three years. These four developers were essentially giant silos of knowledge, and
at first the rest of the team did not know enough to be able to break down tasks. Each of
the four knew different areas of the code, so while there was some overlap in their
knowledge their specialization was a serious hurdle. Often, we would go to estimate stories
and find ourselves bogged down in discussion of the code and trying to figure out why story
after story were receiving really high estimates.

The overlapping problems of the team were killing us. The four experts knew what
needed to be done but were having a lot of trouble translating from their mental task list to
estimating the time it would take to complete the story. Our business analysts were running
into the same issue one level up. They were not used to providing feature requirements in
story form, much less breaking down these "feature" level stories into stories that could be
reasonably estimated by the development team. I realized almost immediately the stories
were too big, but you can't force immediate understanding. Like any skill, estimating takes
practice. We went through many agonizing estimation and story breakdown sessions before
we started to get some traction. As we began to build up some historical data this closed
the feedback loop. The team was able to see that we were frequently underestimating and
that this was directly connected to the trouble we were still having with story breakdown.

The large amount of time we were taking to break down the stories was causing another
problem. We had only a few developers who were knowledgeable enough to work with the
business analyst to break down the story, and we needed them to do the actual work. To
make matters worse this fed the the four domain experts' belief that they just needed to sit
down and work through the problem as they coded it the way the had always done it. This
attitude that was unfortunately reinforced by the company as they pushed for a follow up
release.

It was a very frustrating time. The team was still in the process of learning how to break
down stories. The original developers often felt that two of them needed to pair in order to
work on particular problems, and this reinforced the trouble we were having integrating the
new people. If we had had appropriately sized stories we could have more easily divided up
the features and had the experts pairing with the new people. Eventually we did get to this
point but along the way we had many instances where the new folks, including myself,
ended up trudging along on our own and wasting time with issues that could easily been
cleared up by the more experienced developers. When we began to get the stories closer to
the right size from the business analysts it eased the tension between preparing the work to
be done and actually doing the work.

In the end we had to convince the business that pairing in the new team members and
breaking up the pair of original developers was necessary. By taking some time now to pair
in the new team members we would perform better in the long run, even if it would slow us
down in the short term. I can't say there was any one thing that convinced the four original
developers to start pairing more and helping to transfer the knowledge. I think it was a
combination of things. One was simply getting to know the new people better and building
team bonding and chemistry. Another was the hard work of the new people, especially one
of our team who went on to take a leading role in the implementation of a complex feature.
Lastly I'd like to think I had some influence as I reinforced what they already knew but just
hadn't accepted; that in order to achieve the benefits of agile we had talked about they
needed to change. If the team was to meet the challenges the business was issuing, change
was a must.

We continued to have problems with estimation. The combination of the lack of
experience, vague task definition, and low team buy in continued to drag us down. Lots of
cards were started; few were finished. We started finishing cards, but found that estimates
of hours often took weeks. Stories would spawn additional stories in order to reach
completion. For example, a story that started with a two day estimate would generate a
week's worth of stories (sometimes even more than a week!). Many agile developers would
say "Great! That's part of agile." Gaining clarity on a story and understanding that there is
more to it is part of the process. Unfortunately this was happening far too often. We needed
to translate business requirements into stories and break down these stories into tasks the
team could more accurately estimate.

Are We Agile Yet?:

Despite some agreement from the business that we needed to invest time to change the
team, they continued to be concerned about the timeline for the follow on bug fix release.
They thought that because we were agile now they would get frequent releases, not
understanding that the transition was far from complete. Here was where we got into more
tension between keeping development moving and converting the team to be agile. It's one
ting to declare agility; it's quite another to be a fully functioning agile team. I would argue
many teams never make it and other people would argue that perhaps not every team
needs to. That is a topic in and of itself. In our case one thing we were striving for was
frequent and meaningful releases. Unfortunately at this point we weren't really ready for
either. The legacy code base was making it difficult to turn around features in a weekly time
frame. Even if we did, we did not have the infrastructure in place to accomplish a weekly
deployment. A week was a difficult to impossible timeline. The developers manually put a
new release out on the preproduction environment, and the QA team ran their manual
regression testing. Our team was faced with a choice: spend time improving our team or
spend time on the release.
Our next battles revolved around planning, estimation, and definition of done. We were just barely managing to put together enough stories with enough clarity actually to plan an iteration. We continued slowly to improve on the estimation front but were really held back by stories that were too big. This was a double edged sword: since our estimation skills still needed development it was harder to discover just how big some of the stories were. Couple that with poor story definition and we spent an inordinate amount of time in estimation for a number of weeks as we fixed these issues. On the story definition side, we carved out some time ahead of planning sessions to vet stories with domain experts. We tried to break down cards as much as possible before estimation. On the estimation side we finally started getting actuals that more or less correlate to amount of work spent on the card. This allowed a synergy to form that got us away from the epic planning and estimation sessions.

We needed to cross train but the business still wanted the best people working on the task. The new team members were not getting enough exposure and the code was complex enough that without a lot of exposure it was difficult for the new people to contribute. At this point one tactic we employed was to start pairing the new hires with the domain experts in the different areas of code. While at first it was slow going and met some resistance, the new people managed slowly to gain knowledge and were more able to help. Some of the pain of only a few developers having all the knowledge started to diminish, but it was not the last we would see of it.

Four months after the start of the transition the entire development team across all projects participated in a multi-day training event. It was intended to help build consensus, understanding, and acceptance of agile practices outside the day to day environment.

On the first day we paired the developers up and worked through an exercise on testing comparing two code bases that did exactly the same thing: one written by an individual with no tests in a very procedural, non-object oriented fashion; the second test driven with objects. We asked the developer pairs to add a feature to both, adding tests to prove the new feature worked. All but one of the pairs failed to add the feature to the untested code, all pairs succeed in adding both the test and the feature to the second test driven code. This exercise opened some eyes, but the group as a whole was far from convinced. Could these same results be replicated in their day to day lives with a much larger legacy code base?

On the second day we ran a simulated project complete with product owners and iterations. Each iteration was roughly an hour and began with an abbreviated planning game, followed by coding. The development group was divided into four teams of six to eight developers and each had a product owner in the form of an agile coach or senior agile developer. This went much better than planned: most teams got through all the cards to build "the game of life" and one even got through the extra wishlist items, fulfilling a suite of unit and functional tests along the way. They saw the value in having a suite of acceptance level tests that let them know when they were done.

After the training there was a definite upturn in the team’s opinion of agile. It was a local maximum in terms of the transition but it was not the turning point. The main trouble remaining was getting the team members who were least experienced from an agile perspective the support they needed to reinforce the process. They were willing to test but often saw it as still taking too much time if they could not easily envision a way to test a particular section of code.

They were beginning to get on board with creating acceptance tests to help define what a task’s goal was and serve as regression down the road, but only with people helping craft these tests. This fell to the more experienced agile team members. There were almost enough of us, but we were often still short on resources when it came to balancing between mentoring team members and trying to improve tools and processes to foster a more agile development environment.

The Turning Point:
One critical element to converting a team to agile is gathering enough momentum to change. Two ways to achieve this are having a majority of team members who are motivated to change from the beginning or hiring experienced agilists until they have enough influence and leverage to convert the team. A top down decision without internal team support is unlikely to be successful. In the end, we added three more developers to the team, all with some amount of agile experience. This brought us close to equality based on agile experience. The new folks provided much needed resources and an opportunity to cross train the new members of the team, introducing more consistent pairing to the team.

The best part of the new team members was they were expecting an agile team so they adhered to the practices and actively supported them within the team. This support was key on many levels. They sought to pair because they didn't know the domain and needed the assistance of the domain experts, so they sought them out. When something was not right they were willing to say so. Devoid of preconceptions, they were willing to try things the other developers would not have. They immediately picked up the acceptance and regression testing torch as they went about learning the domain and system. They helped build out the continuous integration server and other agile support mechanisms for the team that had been suffering from lack of resources.

This was a big turning point for the team. New blood and better balance allowed us to provide some social pressure to do more testing, pairing, and evolving as a team. Suddenly I was not the only one advocating for items to improve our development practices. The team started taking the first steps towards policing itself.

The addition of another senior agile developer was the last piece of the puzzle. He tipped the scales so that the experienced agilists comprised more than half the team. After this, I transitioned to one of the other teams feeling confident that while this team still had many challenges ahead they were well on their way to being agile and no team member would go back to the way things used to be.