Abstract—Adopting agile practices is often a long process and one for which there is no existing formula for success. Some common patterns are beginning to emerge, but many of them focus on adoption within larger organizations. While we went through the process of adopting agile practices at IASTA, we experienced challenges that seemed to be somewhat unique, or at least more common, in a small, organic growth organization. Finding advice and guidance on these challenges that could be implemented in our environment from the popular resources was challenging. This paper tells the story of our agile adoption process in hopes that other organizations in similar situations can learn from our experience.

Keywords—agile adoption; scrum; scrumbut

I. INTRODUCTION

Over the past two years (since June 2009) we have been working at implementing agile practices at IASTA. IASTA is a small SaaS software company that was started by three individuals in 2000 who remain the owners to this date. The company has grown to nearly 50 employees and just over $9 million in revenues in 10 years. The founders followed a complete organic growth model, never taking on any outside investment or venture capital. While we were moving the development organization to agile practices, we consulted many books and resources on implementing agile practices, but the vast majority of them were written from the perspective of a larger organization. We noticed our organic growth culture seemed to present some interesting problems. Popular guidance advised starting with a single small team, however, they seemed to assume that you had some other resources at your disposal. These “other resources” were not readily available in our environment. This paper is intended to share some of those problems and discuss some of the ways we attempted to address them so that others in similar situations can learn from our experience.

II. SETTING THE STAGE – WHY AGILE?

Before looking at problems encountered while transitioning to agile, it is probably worth examining why we were transitioning to agile in the first place. As the company grew, we faced problems attempting to grow the development organization along with it. We had issues recruiting talent, onboarding new employees, managing technical debt, tracking progress and meeting deadlines. We also had a lack of product ownership. For the first five years of the company, one individual did all development. That individual did a great job and managed to be able to hear and understand requirements with very little documentation. As the company grew, we reached a point that the throughput of one person just wasn’t enough, so we hired more developers. Several developers were brought on in hopes of replicating the success of the initial developer. Unfortunately, finding developers who could take a project with very little direction and deliver complete working software in a timely fashion did not prove to be easy. Projects ran much longer than expected and what was delivered was significantly different than what was expected. Not only that, but now that there were multiple people working on separate projects without much visibility into when they would be finished, timing releases started to become very difficult.

An obvious problem that was happening is that the developers were receiving very little direction. They were expected to figure it out as they went along, but none of them had ever worked in an environment like that. To address this problem, we began trying to put more process around defining requirements. The result was that large amounts of up-front design were done. The developer(s) took that design and began working on a layer at a time, as they were used to doing. Completing architectural components first meant there wasn’t much working code to show until everything came together at the end. Development would continue working, but since code was completed a layer at a time rather than a feature at a time there was no clear indication of how far along the project was. Additionally, the project generally ran behind schedule so functionality would get cut which rendered much of the already-completed architectural work either unnecessary or incorrect.

It was around this time that we decided to start trying more agile practices. At this point, we had obviously overshot the target and were doing too much up-front design, but we also needed to ensure that development was being given a bit more direction with more frequent inspection points than was happening previously. Visibility into how close to done a project was and when it would be complete was also a major issue that we needed to address. We saw answers to all of these problems in agile, specifically scrum.
III. ITERATING TO AGILITY

A. First Attempts

No one on the team had ever worked on an agile team before. In fact, no one on the team had ever really worked on a development “team” before. Most of the individuals had been used to programming individually. I had just started researching agile development practices and wanted to pick something simple and not very disruptive to try first, so we decided to start with doing iterations. Our plan was fairly simple; we would start making commitments as to what we could complete in a set period of time. Initially, we decided to try four-week sprints. Some of these sprints were more successful than others, but I don’t recall a single sprint in which everything that was committed got delivered. Four weeks proved to be too much time. It allowed us to bring fairly large items into the sprint and then we would pretty much follow the same process as before. The work was broken up, all the back-end architectural work was completed first and nothing came together until the end of the sprint. Often, nothing even came together by the end of the sprint.

In hindsight, probably the worst aspect of the four-week iteration is that there was too much time between inspection points. When the first sprint didn’t go well, we thought maybe it was because this was a new process and decided to give it another sprint. The second sprint went better, so we thought maybe we were getting the hang of it. The third sprint was about the same as the second, maybe slightly better, but then the fourth sprint was a disaster. We had a couple of sprints that went alright, so it was easy to make some excuses and pass it off as an anomaly and go on to the next sprint. At this point we were now six months into the project and time boxing was pretty much all we were doing. We were not using user stories or ensuring there was something of value to the client at the end of each sprint, so a sprint may have been successful, but maybe it was all back-end work that was completed with nothing really valuable to the user and nothing fully integrated yet.

All in all, we kept with four-week sprints for a full release cycle, which wound up being just about a year, ten sprints. That release was originally planned to take six months with a release at the end of 2009. The release didn’t actually happen until the end of May 2010. While it had become apparent that iterations alone weren’t going to help much, it also became obvious that four-week sprints were too long. First off, as I mentioned before, they didn’t allow for very quick change. Also, it was obvious that the team was struggling to determine how much work could fit into a four-week iteration.

B. Take Two

After wrapping up the release described above, we made a couple changes for the next release cycle. The first change was switching to two-week sprints. There was quite a bit of resistance to shorter sprints, but the improvement was noticeable as early as the first planning meeting. There was much less uncertainty leaving the planning meeting. When planning four-week sprints, as tickets kept getting added on, the mood drifted into one of uncertainty and guessing. That transition didn’t happen when planning a two-week sprint. In fact, right when the team started feeling uncertain was precisely when they decided that was as much as they could commit to delivering.

Not only was the change evident in the sprint planning meetings, but also in the sprint review. The percentage of time that the team delivered everything they committed to drastically improved. There were still sprints in which this didn’t happen, but we achieved quite an improvement over our experience with four-week iterations. The fact that we were able to fill the first several sprints with smaller, better-defined tickets definitely helped this along and helped build team confidence as well.

Following two-week sprints allowed us to get through the next, significantly shorter release cycle without any major issues. We had planned four two-week sprints for this quick release and all the top priority items that were desired in that release made it out by the originally planned date in August 2010. As the team built confidence and made it through the smaller, well-defined enhancements and bug fixes, it started to become apparent that the next bottleneck was really outside the development team.

C. The Product Owner

The way we had determined the priority of items in the backlog had always been a bit ad-hoc. As is often the case, the squeaky wheel got quite a bit of grease, but now that the team was making steady progress through the tickets and more often than not delivering on their commitments for each iteration, it became apparent that our ad-hoc backlog prioritization model was not working very well. Executive leadership would often decide what major areas of functionality should be in each release, but the other items in the release were always a bit trickier to deal with. Releases tended to consist of a smattering of tickets from various areas of the software. These were often direct requests from clients and often weren’t related to each other in any way. Because of this, the development team started to complain that they didn’t see the big picture. They didn’t understand why certain tickets were a priority and others were not. We had also had some issues with prior larger projects that lead us to the same conclusion that we really needed to get a product owner more involved in defining features and prioritizing the backlog.

Yes, that’s correct, up until this time we had been implementing agile practices without having a defined product owner. To complicate matters even more, we only have one development team, but we have five products. Despite multiple attempts, I was unsuccessful in convincing the executive team that we needed a single dedicated product owner. In an organic growth company, you are always a little behind the eight ball. You are completely funded from incoming revenue and thus the typical growth model is to initially find ways to absorb the work into existing positions instead of creating new ones. So, instead of getting a dedicated product owner, we created a Product Owner role for each product and asked the person from the support group that was most familiar with that product to fill that role.
in addition to their current responsibilities. While this was obviously not optimal, it was still a major improvement over what we had before.

We rolled out the Product Owner role in September 2010, so we now had one development team consisting of four developers and one quality assurance engineer with five product owners. That was one product owner per development team member. The obvious first problem was that we really needed the development team to be working from a single product backlog. We accomplished this by implementing a release backlog in addition to each individual product backlog. The release backlog became the single backlog from which the team worked.

D. The Release Backlog

Since we had five product owners, they each maintained their own backlog. However, in order to present this to the development team as one backlog, we would get the product owners together as a group and combine the top items from each individual backlog into a single release backlog. We would generally have an initial time frame in mind for a release, so at the start of that release cycle we met to fill 60-80% of the available time from the individual product backlogs. We would then hold a release planning review meeting near the end of every sprint to review the progress that was made and ensure that the product backlog was still correct and make any modifications that needed to be made. During these release-planning meetings, the group of product owners would discuss the importance of each of their individual backlog items and we would come to a collective agreement about where it belonged on the release backlog. During this process, we would also attempt to take into account grouping similar backlog items together where it made reasonable sense so that focused sprint goals could be set.

Our biggest issue with this approach was that the Product Owner role was still not the top priority for the individuals who were filling this role. Meeting attendance and availability were a constant battle. We also ran into many cases when it seemed like there were many more people in the room than necessary for a meeting. We often needed several product owners during the sprint planning and estimating meetings so we would have everyone in the room, but each product owner would only be engaged for a short time. This very likely was related to the problems we had getting product owners to be available for the meeting as they felt that they were not fully utilized in the meeting and that their time was more effectively spent elsewhere.

Communication seemed to be another major issue with this approach. With multiple product owners, it was difficult for the team to know which one to approach with which questions. This became especially complicated for core components of the software that are shared among multiple products. Not only did it cause confusion on who to turn to with questions, it also stifled communication and participation in meetings simply because it created a larger audience at the meeting.

Despite the negatives mentioned above, this approach was definitely an improvement over our existing methods. Having individual product experts in charge of prioritizing the product backlog items for that area of the product was much more manageable than trying to prioritize them as a single list. It allowed us to spread the load of managing a very large backlog among multiple individuals instead of having to hire another individual. With a few additional tweaks, this approach probably could have been even more effective. We definitely saw the most success and most effective work when we were able to limit the number of product owners involved in a single sprint. In reality, one person really does need to lead this effort, even if there are multiple product owners. Someone needs to coordinate and facilitate the meetings. I filled that role in our implementation as the development director, but a Scrum Master or one of the individual product owners could easily fill this role. If this individual does a good job and keeps the above issues in mind, this model would probably work fairly well. Of course, if that is true, in reality, this individual is probably the actual product owner. He or she would be the one making the final call on things like priority and resolving disputes between the other product owners. Based on my experience, doing this job well would definitely be a full time job and thus you would actually end up with a single product owner with a good supporting staff of product experts.

IV. ADDITIONAL CONSIDERATIONS

The prior section of the paper focused mainly on process changes we applied incrementally over the course of time. This was definitely a very agile approach, adapting our process as we went to work best for our team. There were also several other areas that needed some attention that were ongoing throughout the process changes already described.

A. Scope Creep and Team Ownership

As I mentioned earlier in the paper, one of our initial issues was that developers were given very little direction. We attempted to counter this by doing a bit more up front planning. Generally, the way this was supposed to work was that an executive summary would accompany any project that involved more than a couple weeks of development. Executive Summaries provided a high-level description of the project vision and some bullet points on what functionality and value the project should provide. Each Executive Summary received a very high level estimate of effort from our architect and was then reviewed and ranked by the executive team. Based on these numbers, we determined where the project fit in the roadmap. When that was determined, the project would go to the product owner to work with the designer to flesh out the details, which would generally result in complete mockups and user stories. These user stories are what would make up the product backlog and get estimated by the development team and brought into sprints.

At a high level, that process is fairly sound. However, we did not do a very good job of getting team ownership of the project from day one, so what generally ended up happening was that the executive team would decide project X belonged in this release given the executive summary and
high level estimate. However, when it would go to the product owner/designer, the executive summary and high-level estimate weren’t always kept at the forefront of the design and the mock-ups and user stories frequently went beyond the scope of the executive summary. Because of this, when the development team finally saw the project and estimated the user stories, the entire project was much larger than what the executive team had planned. To make matters worse, expanded scopes and extended timelines were not always conveyed back to the executive team until the team got further down the road and it was painfully obvious that either the desired release date would have to change or a large number of the stories would have to be cut.

The main cause of this issue was that the original executive summary was not being communicated throughout the lifespan of the project and there was a lack of team ownership in the project. The development team didn’t see anything about the project until after the scope had already been greatly expanded and didn’t necessarily know it was beyond the scope of the executive summary to provide appropriate feedback during estimating. Also, since they got involved so late in the process and weren’t always aware of the overriding goals of the project, they were not encouraged to provide alternative innovative solutions. Increasing the visibility of the executive summary should greatly reduce this problem as well as foster a higher level of team ownership. It would also address the development team’s concerns that they could not see the big picture. Formal chartering and reporting would keep the team and the design connected to the high-level goals of a project. We’re not talking about a massive business proposal or anything like that, but some sort of document listing the vision of the project with goals and success measures. In reality, the executive summary we had was fairly close to a project charter, we just didn’t treat it as such. We should have ensured that the entire team knew and understood the charter for the entire lifecycle. In fact, it should have been posted in the team room right by the information radiator. They should understand the goals of the project, the success measures of the project, and the expected timeline of the project. Getting the development team involved earlier in the project and doing less of the design up-front and a lot more just in time would have helped as well.

B. Technical Practices

So far I have really been focusing on more of the process changes that we made and attempted, but I can’t downplay the importance of some of the technical practices we implemented during that time as well. When we started this transition, we were starting with 8-year-old software that had been written by one individual. There had been no automated testing and most of the code was very tightly coupled without much object-oriented design. Individual developers were working in their own branches of the code for long periods of time and then attempting to merge all their changes back into the trunk just before release. There were no dedicated QA employees; our support group did testing without any pre-planning or documentation on what exactly would be tested. This situation obviously caused many issues. These practices affected some of the decisions we made along the way. Without changes we probably would not have been able to do less than four-week sprints at the beginning even if we wanted to.

As we began to implement some agile practices, we also changed some of our technical practices to be more agile as well. One of the first things that changed was that the developers started working in the same development branch and updating frequently. This alleviated the massive merge that used to occur at the end of a release cycle. We also hired some full-time QA engineers. We started implementing better object-oriented designs, which caused the code to become less coupled. All these changes enabled us to shorten our iteration to two-weeks.

Writing less-coupled code allowed us to introduce unit tests. We still did and still do struggle with unit testing because we are using an older development environment that does not have built-in support for unit tests and there is still very limited unit testing we can do on the legacy code when we modify it.

Another technical practice that we changed was automating the build and deploying to a development environment twice a day. Now that the build was automated, we could automate the unit tests to run with every build. Finally, we setup a continuous integration server that would build the software every check-in and run the tests.

These changes to our technical practices played an integral role in the level of success we were able to achieve with agile practices. In most cases, we weren’t able to make the next leap in our agile adoption process until some technical changes were completed. Some current technical practices are still cause for our largest pain points. Continuing to improve technical practices will help us continuously evolve our agile practices.

V. AN OUTSIDE PERSPECTIVE

After almost two years of trying to get agile practices implemented with limited success, we finally got approval to bring in an agile coach. I had attended a presentation on scrum metrics given by Scott Downey and Dr. Jeff Sutherland at Agile2010. Scott seemed like a great fit with years of experience bootstrapping many, many teams, so I contacted Scott and arranged for a full month of on-site coaching broken into two or more sessions. The first session was limited to just over a week due to other obligations he had, but the effect was immediate.

One of the first things he noticed was that the level of communication on our team was nowhere near the level that you would usually see on an agile team. Scott’s suggestion was that a large cause of this was simply the size of the audience in most of the meetings. At the time the team consisted of 4 developers, 2 QA engineers, the Scrum Master, a designer, and a technical writer. There were also the multiple product owners in the room as well as myself in most meetings. The obvious first person to eliminate from the meetings was myself. As the designer and technical writer had limited involvement in the delivery itself, they were also moved outside of the team and became resources for the product owner to manage. He also strongly suggested.
we re-think our multiple product owner approach and go with a single product owner.

Given that most of our product owners were already over allocated with other responsibilities, we decided to move forward with a single product owner. We found an internal resource who had previously performed some of the tasks of the product owner and who understood both the software and users very well. This reduced the number of product owners from five to one and gave the development team a much less confusing line of communication. It also greatly reduced the number of people required for meetings, which greatly increased the amount of interaction. Offloading all other responsibilities enabled the product owner to fully dedicate herself to fulfilling that role.

Ultimately, the arrangement of multiple visits with time between them is working very well as it is giving us time to digest what we have learned, act on some of the recommendations, and have another chance to get more feedback and guidance. During Scott’s first visit we focused a lot on getting a better understanding of estimating. We went through the process of anchoring our story point scale, a key step that we had previously overlooked. We also defined “done” more clearly and educated the new product owner and team on the rules of getting backlog items accepted so that they count toward the velocity. After a few weeks of practicing those things on our own, Scott returned and we spent a bit more time on the rules of getting backlog items accepted as “done” and on how to adopt work during a sprint. We also started to work on focused backlog grooming and how the team can help with that during a sprint.

VI. CONCLUSION

It has been a lengthy process and we still have plenty of room for improvement, but we have seen a lot of benefit in undergoing an agile adoption. When we started our transition, our first release missed the target date and we still didn’t have insight into how close to done we were. We kept pushing back the date a month at a time until we finally released five months late (almost double the initial expected length). Obviously, our customers did not appreciate this continual adjustment. While our current release cycle has also nearly doubled the expected length, we were able to identify this early in the release cycle, before any dates were communicated to our clients. We were then able to use the data we had to adjust the expected release date to a more realistic date. This single adjustment proved to be accurate and we were able to confidently communicate a release date to our clients that did not need to be adjusted.

So why is agile adoption at a small, organic growth company any different than anywhere else? One of the biggest challenges in an organic growth company that is commonly less severe in other environments is the scarcity of resources. Every company is challenged to watch costs, but in a small company you have a smaller number of people and thus a smaller amount of experience and skills to draw from versus a large organization. The nature of organic growth tends to keep you one step behind. You are financed off your revenues that are 100% re-invested in the company, but often those revenues are coming in after you deliver the product and services. Generally existing staff must absorb new responsibilities for a while before dedicated resources are made available to perform the tasks associated with those responsibilities. Because of this, everyone is almost always wearing multiple hats and stretched pretty thin. That definitely describes the situation we encountered with the product owner role. The obvious solution to our problem at that time was that we needed a dedicated product owner. In fact, many books out there say it is crazy to even attempt to start an agile transition without a dedicated product owner, but it just wasn’t possible at the time. It was not yet a proven need that would, without a doubt, make an impact on the bottom line. So as with any new responsibilities in that sort of situation, you make do with what you have by starting to perform those tasks with current resources until you reach the point where the tasks have provided obvious benefit but the people doing them are overworked. At that point, if the person that stepped up and took on that role is doing a good job and wants to do that full time, you back-fill to cover his or her previous responsibilities. Otherwise, you look for new talent that has experience performing the new set of responsibilities.

The most valuable lesson we learned in this process was that there is no replacement for experience. We ran into resource constraint problems on this as well. Had we brought in an experienced agile coach like Scott sooner, I’m sure our adoption would have been much smoother and quicker. In fact, if you refer to Figure 1, you will see how the team velocity grew both before and after we engaged with Scott, starting with when we implemented two-week sprints. We were obviously making improvements and improving our velocity, but the rate of improvement went up quite a bit with the experienced guidance Scott brought to the table. However, when resources are tight, the cost of an agile coach is difficult to justify, especially if you have never utilized that form of consulting before. Ultimately, the main thing you are paying for with a coach is to have an experienced person there to guide you and answer questions. You don’t really know how fully utilized they will be until they arrive. Maybe management wonders whether you will have enough questions to keep them busy and is afraid they will sit around playing solitaire all day? Chances are if you are considering bringing on a coach, this will not be a problem. You can always address this concern by preparing a list of questions that you think the coach will resolve. It won’t take long for that list to take up multiple pages.
The other challenge we faced that wasn’t addressed in the resources I consulted was simply the fact that we were a small company. Many books suggest that you start implementing agile practices on a single team so you can learn and show success before championing the adoption process across the organization. We only had one team, so it was either start or not start. This coupled with generally scarce resources means you should plan for a long, gradual transition. If you don’t already have more modern development practices in place like test-driven development and continuous integration, then it will be an even longer journey, as you will need to make those changes as well. As with any long process, you should be aware of this from the start so you can prepare to do some change management. Also, ensure that everyone understands it will be a long process. If you go in saying “We’re going to move to agile tomorrow and it will be great”, and then it turns out to be a long process with varying degrees of early success, you have set some faulty expectations and will lose trust in the organization when those expectations aren’t met. In the same sense, you should be careful about what you call your process in the early stages. If you say “We’re doing scrum” but you aren’t completely doing scrum and you aren’t having the best success right away, the process could get a bad reputation with management when it isn’t actually the fault of the process. There is a lot of talk currently about scrumbut and how bad that is. I would argue that having a partial implementation of scrum (which is what scrumbut is, “we’re doing scrum, but...”) is only a bad thing if you are planning to keep it that way. In a small, organic growth company, you won’t have the time or resources to just flip a switch and be doing 100% pure scrum day one. You will spend a decent amount of time in scrumbut. However, as long as you realize that what you currently have is a scrumbut process and have concrete plans to transition into a full scrum implementation, scrumbut isn’t that bad. In fact, I would prefer to call that not-scrum-yet rather than scrumbut.