Back to Basics: The Role of Agile Principles in Success with an Distributed Scrum Team

Steve Berczuk
FAST
steve@berczuk.com

Abstract

Agile processes rely on feedback and communication to work and they often work best with co-located teams for this reason. Sometimes agile makes sense because of project requirements and a distributed team makes sense because of resource constraints. A distributed team can be productive and fun to work on if the team takes an active role in overcoming the barriers that distribution causes. This is the story of how one product team used creativity, communications tools, and basic good engineering practices to build a successful product.

1. Introduction

FAST is a search company with its headquarters in Norway. In the fall of 2005 the company made a decision to build a search application built on top of the company’s core search platform. Because of a desire to leverage the respective teams’ areas of expertise, to facilitate knowledge transfer, and to support the international nature of the company, the application team was split between people who were in the Boston office, who had experience in application development, and people from one of the Norway offices who had experience in the core search technology.

The team started out in a co-located office space for a number of months and as the project progressed members of the team who were from Norway moved back home and continued to work with the team remotely.

The project’s management committed to using Scrum as a development method, since requirements were changing rapidly and the team needed to deliver quickly. Management support of the agile process included enabling the team to embrace, adapt, and manage the process so that it worked better for them rather than mandating certain styles. While management commitment is helpful in the adoption of an agile process, it is not enough. In implementing Scrum the team faced challenges managing the product backlog and coordinating between two offices. Scrum practices and values established when everyone was in one location helped the team to develop a work style that made the transition to distributed development easier.

In this paper I’ll discuss four lessons we learned from this experience:

• While distributed teams can make agile difficult, the distribution often amplifies existing process issues rather than being the issue itself. Following the agile process by the book before attempting customizations can get a team on track.

• Management support for letting the team find its way (with guidance) is a major success factor in any agile practice, more so when the teams are distributed; When the team embraces the process and works to improve it, the team is more likely to be successful.

• Good engineering practices are essential to having an agile project succeed. The benefits are even greater when the teams are distributed.

• Agile is about people, but distributed agile requires good tools to help people communicate effectively over distances.

2. All Development is Local

Agile methods rely on frequent, good quality, feedback to facilitate the ability to change direction as business needs change. This is what makes “agile” agile. Team distribution always presents challenges. Communication and interaction is so essential to agility, the distance between team members amplifies communications deficits and lengthens the feedback loop when something goes awry, making the risks to agile teams greater. Traditional project methods rely
more heavily on artifacts (sacrificing speed, and perhaps productivity). Agile methods have fewer artifacts, and rely on frequent feedback to ensure that the team is doing the right thing.

The organization’s decision to use Scrum provided the framework to enable the team to meet the demands of the product owner. Still, early on the team struggled with working in a rapidly changing environment without having the right change-management processes in place. The team and the product owners needed learn to work within that framework and understand the basic elements of Scrum.

Since the team started out co-located we had a chance to develop a shared model of the Scrum process before we needed to address the added complications caused by distribution.

While each team is unique, and there is a great temptation for teams to take a process and start by customizing it to their “special” circumstances, teams are better served by starting with the “text book” and seeing where things break and then make changes to the practices that align with the values of the process.

Scrum has only a few rules and by nature of being an agile process, the planning horizons are small enough and frequent enough that the team can adapt and try new ways of executing the basic process.

The elements of Scrum that our team needed to learn to work with were:

- The Product and Sprint Backlog
- Iteration Length
- The Daily Scrum Process
- The Sprint Review

Having a common understanding of how these elements work when the team was co-located made it easier to address the complications of having a split team.

While the agile principle of adapting to change and delivering frequently were something the team understood at the outset, the team did not initially embrace the value of having a well defined sprint backlog. The team also appreciated the role of basic engineering practices in facilitating collaboration and communication.

2.1 Recommendations

- Ensure that all team members understand and embrace the values of your agile method. Work with the remote team closely to ensure that they have the feedback that they need to make the method successful.
- Avoid the temptation to customize the method immediately. Have a couple of

3. Managing the Backlog

One challenge that the team needed to address while they were still co-located was one that all teams have regardless of where people are located: taking control of the product backlog.

In Scrum (and most agile methods) a key success factor is having the team commit to delivering something in a time-boxed window and accepting new work. As is typical in many new projects that need to prove themselves to a market or a customer, early on the team had difficulty meeting its deadlines without heroic efforts. Part of the reason for this was that the contractual and market requirements were ambitious. The larger part of the reason was that these ambitious goals did not make it to a well managed product backlog.

Early in the project the backlog items were not well defined so it was difficult for the team to know what they were committing to. After a sprint started, new requirements would come in and the team was faced with an every growing list of work, while still committed to the time box. Because of the amount of work, the team was reluctant to spend time as a group to understand the requirements. Team ownership of deliverables is a key part of Scrum. Early on there was a tendency to shunt certain features to certain team members based on expertise made for a situation where similar things were done in different ways throughout the application. This demonstrated the affect team structure can have on architecture.

By following the Scrum process for sprint planning we addressed these issues successfully.

3.1 Understanding the Workload

In most cases product owners don’t intend to give teams an unreasonable amount of work; it’s to everyone’s benefit to have a workload that the team can commit to and deliver successfully. The problem occurs when the product owner either does not understand the effect of a change or when the team simply agrees to the change without enough analysis.

One of the first things our team did was to track progress against a backlog and generate burn down charts. Even in the absence of precise requirements (or at least user stories) and accurate estimates the burn down chart provided a tool for indicating that work was added to a sprint. When the team did not
meet a sprint goal, we could point to the burndown chart for the sprint and indicate that there was an upward spike midway through. Once we could demonstrate this, the product owners got into the habit of introducing changes with a sense of relative priority. We now could anticipate before the end of the sprint if we would miss the goal. The burndown chart provided a basis for a conversation about the costs of introducing change into the project mid-iteration.

3.2 The Virtual Wall

When the team was located in one room we used traditional backlog tracking tools including index cards on a wall and a burndown chart generated using a spreadsheet approach. The team liked the visual feedback that this provided.

Figure 1 shows the state of the wall during one iteration. We also had clocks for each time zone the team was in. While simplistic, the clocks served as a reminder of when people were available and helped to connect the teams.

As the team experimented with different tools we found that we keep returning to the index cards on the wall. People like the visual feedback, and it gives us a good reference point for discussion during the daily scrums.

The index card approach initially had a couple of problems for us. While people liked the visual feedback, we initially had a backlog with many tasks that were too small, and some team members had a hard time remembering to update this board, and the task of generating the burndown chart by hand took up a significant amount of time.

Once the team was split between locations not all team members could update the index cards (and having a proxy did not work).

Over time we experimented with a couple of tools to solve these issues. We tried XPlanner and Jira and settled on Jira. A web based tool has the advantage of allowing remote team members to review the backlog and update tasks wherever they are. Likewise when our product owner traveled frequently he could review the backlog easily.

As of now, the team has not only accepted Jira, but has embraced it, seeking ways to use it more effectively.

We now use data in Jira to track burndown in terms of effort remaining. Our team is now mostly colocated and we generate index cards from the task descriptions by exporting the descriptions to a spreadsheet and using a print merge application to generate the cards. The cards give us visual feedback on number of tasks left and roadblocks, and also provide us with a reference during the daily scrum. With this approach we can display progress with both remote and local team members easily in our office and also allow everyone access to the backlog online wherever they are located.

Figure 1 The Scrum Wall

3.3 Recommendations

- Tracking burndown is essential to realizing the benefits of Scrum. Keeping to a backlog is not about saying “no” the customers. It’s about understanding what the obstacles are and embracing the reality that things have priorities.
- When you have distance to contend with technology can help. Wikis and issue tracking systems allow people to synch up regardless of location. Having a tool to track backlogs and burndowns is essential for a distributed team, and the tool will be most successful when the team embraces it and is willing to adapt it to their needs.
- Don’t ignore the value of low tech tools, even if only one location uses them. Things like clocks showing the other time zones help the teams to feel connected.

4. Iteration Length

The application the team was developing had one key customer that we needed to be very responsive to. The team started with the “canonical” 4-week sprint. Almost immediately we had to address how to deal with frequent customer requests that arrived during the sprint and required a fix before the sprint started. This meant that in practice, the sprint backlog was fictional. The burndown chart helped us address the problem of
explaining why the sprint goal was not being met, but it was still not good for morale to not meet the goals. Strictly sticking to the backlog was not an option because our goal as a team is to respond to customer needs. Terminating the sprint also wasn’t appealing at first since we had so many interrupts we would have stopped progress. We needed to find a way to use Scrum to improve progress. Interrupts during the sprint interrupted productivity and flow.[1] The Scrum Master negotiated with the product owners to decide if shortening the iteration length would help. We decided that in most cases a 2-week iteration would strike a good balance between how frequently we needed to (possibly) deliver and how much time was needed to accomplish a significant task. We still allocated time during the sprint to address critical issues, but this time became smaller and we could more reliably limit “releases” to the sprint boundaries.

By using smaller sprint durations we were able to maintain a stable backlog while still being able to release as frequently as we needed to. As a side-benefit, the shorter sprints allowed to validate that team members in all locations were all aligned by forcing us to re-plan frequently. Longer sprints encourage the tendency towards isolation.

### 4.1 Granularity

When we moved to two-week sprints some members of the team had difficulty conceptualizing of work in two-week sizes, much less having smaller subtasks. A frequent complaint was that some work could not be broken down into two week sized parts. This view is based on a faulty understanding of the role of design and rework in an agile team.

Agile methods are about delivering value the end of every sprint so that the stakeholders can evaluate and change course. While some features might not be able to be done completely in two weeks, it is always possible to define some user visible increment that gets you on the way. There will be rework as we discover things about the feature and the implementation, but if the team is following good engineering practices, the rework should be easy and the team and the product owner will learn from the experience.

### 4.2 Recommendations

- Shorter sprints make sense whenever you have rapidly changing requirements, even if your team is co-located.
- The problem with distribution is communication bandwidth, so err on the side of shorter sprints to force group checkpoints where there are concrete results.

### 5. Engineering Practices

Engineering practices provided the feedback and quality levels to allow the team to deliver more reliably, which reduced re-work and improved confidence that we could deliver on time. Good engineering practices went hand-in-hand with the shorter sprints, since if we needed to be able to ship every couple of weeks we could not afford to spend time on pre-ship sprints. In practice we did not have to ship every two weeks, and it took the team a while to be comfortable shipping without a ‘pre-ship sprint’ but the having that goal allowed the stakeholders to feel more comfortable with letting the team work with a fixed backlog much of the time.

Much as distribution magnifies the effects of communication gaps, it also provides a way for the value of good engineering practices to shine. The engineering practices that had the most benefit to the team were:

- Unit tests, which were a way to communicate architecture and the status of the project.
- The continuous integration build and the “no broken builds” discipline allowed any team member to start work on something without relying on the other team being in the office.
- Integration tests using Quick Test Pro allowed the team to detect possible problems quickly so that they could more easily be corrected.

As quality improved the team was better able to avoid interruptions caused by a need for urgent fixes, thus providing for a happy feedback cycle of quality needed to maintain the ability to ship quickly, while at the same time reducing the need for unplanned releases.

In addition, we realized that engineering practices such as unit testing and continuous integration had even more benefit as the team spread out. While a module with few unit tests or a broken build is always problematic, the problem was easily resolved when the responsible person was across the room. When that person was across the sea (and perhaps asleep when we needed information) the effects of not following these engineering practices were greater.
5.1 **Recommendations**

- Unit testing and continuous integration are essential for distributed agile to work as they provide a communication mechanism.
- The quality that good engineering practices provide can give the team more room to work out issues that distribution causes.

6. **The Daily Scrum**

While it should be obvious from the method’s name, the central process that was important to follow was the Daily Scrum. While all of the other aspects of Scrum translated pretty clearly when the team was distributed, the daily scrum, by its nature presented challenges.

At the beginning of the project we struggled a bit with following the “3 Questions” format in part because the backlog didn’t lend itself to small enough tasks, and also because we had to overcome the “status reporting” culture. As Scrum Master I needed to discourage the team from reporting to me in favor of sharing knowledge with the rest of the team.

Once the team distributed the one challenge that we very specific to the distribution of the team was how to run the daily scrum and the planning sessions.

Keeping to the daily Scrum format was especially important once the team split between locations, as it was the one time we gathered as a team. There were a few issues we needed to overcome. First because of the time difference, while the scrums were at the star of the Boston team’s day, they were close to the end of the Norway team’s day. This led to a strange dynamic in terms of the “3 question” format as some of us were talking about “today” and others “tomorrow.” Also it was difficult to find a time that worked for both teams.

We settled on a time that was a bit earlier than the Boston team’s ideal start time, and close to the end of the Norway team’s day. The Norway team found it useful to have a mini-scrum of their own at the start of their day, which reinforced the value of the scrum, but also brought home the point that co-location has benefits that are hard to reproduce even with good tools.

Also it took the team a while to find a good medium for the scrums. We started with a telephone conference line and a speaker phone. Often a speaker phone made it difficult to hear. Skype using speakers and a microphone worked better. It’s not clear that there is a technology solution that enables truly high-bandwidth communication. Though having spent the time together was very helpful. [2, 3]

One other logistical issue was office space. The Boston team met as a group and the Norwegian part of the team dialed in to the scrum from their desks. This made it very important that the team follow the format and be concise, but the team also needed to be careful to address their comments to the microphone, which had an affect on the dynamic; we had to work hard to maintain a theme of “sharing” rather than “reporting” during the daily scrums.

Remote daily scrums have challenges. In many ways it seemed to work best when the remote part of the team had something separable to work on creating a “scrum of scrums” structure. Keeping the scrums short and on target was the most important way to keep remote parties engaged.

6.1 **Recommendations**

- While some people will have more of an interest in tools than others, encourage the team to participate in any decisions regarding tools you use to facilitate communication.
- Regardless of tools, agile development is a people-oriented activity so having the team physically co-located for a brief period — at least 1 sprint — early in the project helps.

7. **Sprint Review**

The Sprint Review was the one aspect of Scrum that the team took the most liberties with, and it and once we went “by the book” we found that we made major leaps toward having shippable code at the end of the sprint.

The team marked the start and end of the sprints by having a meeting where we discussed the progress, had mini-retrospective sessions where we discussed what we did well and what we could improve on also discussing the review items from the last sprint review.

For reasons related to work being incomplete and a reluctance to show anything other than success we did not always have a demonstration at the end of the sprint. Other teams at FAST were in the habit of shifting the end of the sprint when work was complete. The problem with this approach is that it makes it difficult to truly evaluate the state of the project.

Following the review process gave the team and the stakeholders the benefit of very high bandwidth communication on the status of features and provided an opportunity to discover when there was a gap between feature requests and implementations, in spite of a good communication ethic. Even though the
demonstrations sometimes didn’t always work as planned they had the effect of increasing the confidence that the stakeholders had in the product team.

A side benefit of enforcing the demo process was that the team and the stakeholders rallied around the idea of improving one of the aspects of the product that everyone agreed was weak, yet no one had prioritized for fixing: the installation process.

The distributed nature of the team posed a problem for sprint review. While technologies such as web-based meetings and conference lines allowed for remote participation, time zone differences presented difficulties in scheduling a sufficient block of time that was suitable for all parties. We often compromised by having the sprint reviews center around the local team and the product owner, though since many of our implementation team members (who are key stakeholders) are remote, the tools we used to make distributed development work for us are adding value to engaging all stakeholders.

7.1 Recommendations

- A meaningful sprint review helps the team take stock of where they are; this is more important with distributed teams than co-located ones as less ad-hoc communication occurs with distributed teams.
- The problem with distribution is communication bandwidth, so err on the side of shorter sprints and always have a sprint review with a demonstration and retrospective component.

8. Conclusions

The main reason that it is difficult to do agile with distributed teams is that distribution can reduce communication bandwidth. Co-located teams that don’t communicate well can also fail with agile methods. But the rules of agile methods serve to increase communication and feedback. Any team is best served by following the rules of the agile method with as few adjustments as possible. Distribution increases the damage that non-compliance can cause. If the team feels like it owns the process and the tools it is more likely to be able to overcome obstacles and be successful.

9. References

