Scaling Agile: Finding your Agile Tribe

Erik Moore
Siemens Medical Solutions USA, Inc.
erik.a.moore@siemens.com

John Spens
ThoughtWorks, Inc.
JSpens@thoughtworks.com

Abstract

The ability of agile practices to scale to "large" software development efforts (more than three teams or 30 team members) has been widely debated in recent years. While a variety of inhibitors to scaling agile are frequently cited, this experience report describes how the primary challenge to scaling agile that we faced was finding the right people to build our agile tribe. The effective adoption of Agile requires passionate, motivated individuals who accept change, think like a business owner, operate willingly outside team walls, and identify and drive program issues. Finding such a large number of people with these qualities is much more difficult than establishing methods to support communications, progress tracking, and other agile practices. We conclude by describing the attributes of the type of person that we believe will thrive in a large-scale Agile shop.

1. Introduction

This report describes an experience with a large-scale global development effort (over 300 people across 3 global sites.) We describe our experience with scaling this project and the two key project challenges that we faced: integration and dependency management and continuous integration. While we were able to establish methods and techniques to address these issues, our primary challenge to scaling agile became finding the right people to form our agile "tribe" which led to an identified set of attributes that describe the type of person that we believe will thrive in a large-scale Agile environment. Many talented developers, product analysts, and testers who operate effectively in a small team setting do not fit this profile. This report covers the background of the project, details of the challenges, lessons learned, and conclusions.

2. Project Background

The project under discussion was a large-scale global development effort for the Health Services (HS) division of Siemens Medical Solutions USA, Inc., the healthcare industry's leading supplier of information technology solutions across a broad range of clinical, financial, and management applications and outsourcing and professional services to over 5,000 healthcare organizations in 20 countries. The project team consisted of 300 developers, product analysts, scrum masters, and testers across 25 scrum teams in three development sites: USA, India, and Europe. ThoughtWorks, a global IT consultancy who delivers custom applications, no-nonsense consulting, helps organizations drive agility and creates software, was engaged to provide project management and development support to expand and accelerate the usage of agile practices within Siemens Health Services.

The application architecture followed a Service Oriented Architecture (SOA) approach. The teams used Agile development practices including pair-programming, test-driven development, collective ownership, coding standards, refactoring, and continuous integration.

The project was organized into three sub-programs: core application, common architecture components, and a common supporting application which fulfilled major dependencies for the core application (and other core applications in the HS portfolio) and was shippable as a separate application. The first two sub-programs were developed in the USA and India. The other sub-program was developed in the USA and Europe. The project began with the following number of teams in each sub-program:

- Common Architecture & Infrastructure (2)
- Common Supporting Application (5)
- Core Application (1)

The Common Architecture and Common Supporting Application teams existed prior to the initiation of this project. Over a one year period, the project grew to include 25 scrum teams with an average of 12 people per team (Scrum Master, 4-5 Developer Pairs, 1-2 Analysts, and 1-2 Testers) with the following number of teams in each sub-program:

- Common Architecture & Infrastructure (7)
- Common Supporting Application (7)

978-0-7695-3321-6/08 $25.00 © 2008 IEEE
DOI 10.1109/Agile.2008.43

- Core Application (11)

3. Project Challenges

As the size of the project grew, several challenges emerged that required focused efforts and mechanisms to resolve.

3.1. Integration & Dependency Management

From the onset, we sought to reduce cross-team dependencies as much as possible, allowing each of the Scrum teams to "run" at their fastest pace. However, as the number of concurrent work streams increased, so did the number of dependencies. In particular, we found that adopting a SOA ran counter to our efforts. We quickly faced the prospect of implementing and integrating with a sizeable set of centralized services that would be consumed by multiple project teams. Within a few months, we had accumulated a significant number of integration points, preventing us from delivering fully production-ready software until they were resolved.

Accepting the fact that these dependencies would be part of the project, we sought to create a lightweight and efficient process for resolving dependencies and integration points. We recognized that traditional approaches to dependency management tend to place significant constraints on the sequence of work, reducing the teams' ability to independently prioritize their backlogs. Furthermore, highly contractual, document-centric approaches to defining integration points are work-intensive, prone to error and can create adversarial relationships between teams. We wanted to establish an approach based on direct communication and collaboration. We envisioned that our Scrum teams would:

- Independently manage their integration points
- Employ Test-Driven development techniques and integration tests to define requirements and solidify contracts
- Create cross-team pairs to implement and integrate services.

Our initial integration efforts involving five of the module teams of the Core Application were reasonably successful. After some initial scrambling, the teams self-organized and established a set of procedures for managing inter-module integration.

However, our efforts to facilitate cross-team collaboration within the Core Application benefited heavily from the fact that the five module teams were all closely co-located, within the same organizational structure and employed an active "Scrum of Scrums" model to ensure communication and coordination between the teams. As we pushed this approach out further across the program and the globe, we began to experience significant challenges.

3.1.1. Managing Heavily-Used Services. The Scrum teams within the Common Supporting Application and the Common Architecture Groups provided common sets of services to an increasingly large set of customers. As their customer base increased, so did the requirements to customize, or "tweak", services to accommodate the consumers of these services. Heavily used services were refactored a number of times over the course of the project.

We found that we had varying degrees of success in managing heavily-used services. In those cases where we were successful, it was invariably due to a small group of people from both the consuming and providing teams taking ownership of the services and proactively seeking out future integration points and potential challenges. Conversely, in those situations where such ownership did not exist and people remained within their team walls, we invariably experienced integration challenges: services consistently failing to meet the consumer's requirements and integration points that were poorly tested and failed in system testing.

3.1.2. Managing Distributed Teams. As more and more integration occurred across the globe, we witnessed a tendency to revert back to contractual forms of communication. As always, distributing the development team significantly increases the challenge to directly communicate and collaborate. The natural tendency to rely on e-mail due to differences in time zones frequently led to communication breakdowns and unnecessary blockages – forming barriers to feedback. In some instances, individuals broke through these barriers using instant messaging products, web cams and adjusted working hours to facilitate as much direct communication as possible. At the same time other teams regressed, relying on documentation and establishing an almost adversarial relationship with their integration partners. We were able to identify the quality of the relationship by a few key signs in the sprint reviews:

- The sprint team referred to their integration partner as a single entity, rather than talking about interacting with individual members of that team.
- No one on the sprint team had a phone number readily available for contacting their integration partner or had a member of that team in their IM contacts list.
- Integration points failed during the sprint review demo because no integration tests

were created to validate the services adhered to defined acceptance criteria.

Overall, we eventually recognized that the volume of shared services and integration points was best managed through a service repository and lifecycle management product. The product alone did not resolve our dependency issues: these were resolved primarily by individuals willingly working outside the walls of their team room and taking ownership of the integration effort and responsibility for delivering a fully integrated, fully functional application.

3.2. Continuous Integration (CI)

One of the drivers for adopting Agile practices was to instill a higher degree of quality in the application. With the large number of developers concurrently contributing to the code base, we viewed the adoption of CI as critical to maintaining a high-quality codebase and limiting the accumulation of integration debt. We invested heavily in automated component and integration-level testing and wanted to see those tests run frequently against the most recent build of the application.

Given the size and distribution of our development team and the fact that the program produced a single, integrated enterprise application, the complexity of the build process and the level of effort to maintain it was significant. As the program grew in size, we recognized that the tool set with which we began the project was no longer sufficient. We began experiencing nearly continuous broken builds due to conflation and timing issues on the build server. At this point, we began a lengthy initiative to replace the toolsets and revamp the entire build process using AntHillPro and Subversion. As a result, we created a build process that effectively managed our myriad of dependencies and supported a total build of the application in a reasonable time.

We were, however, less successful in establishing a consistent culture of working builds. We were frequently unable to maintain consistently green builds without dedicating management resources to hound the Scrum teams and track down errors. Initially, we assumed that this was a result of the build issues and that, because it was so difficult to achieve a clean build without throttling check-in activity, it was unreasonable to expect the teams to aggressively resolve problems that were not true defects.

As the new build system came online, we failed to see the passion for producing a working build reemerge. Many members of the Scrum teams failed to monitor the build beyond their own module-level build. Producing fully functional builds at sprint reviews and toll-gate check off points continued to require code freezes and extensive management intercession. While there was a core of individuals who are passionate about producing a working build, we were forced to consolidate that group into an integration team, responsible for ensuring the integrated builds were successful. We were extremely hesitant to take this step, hoping that team ownership of code would drive people to solve integration challenges. However, it was very difficult to expand that passion and sense of responsibility across the entire project team.

4. What We Learned

Over the lifetime of the project we encountered two key challenges due to the scope and scale of this effort: integration and dependency management and continuous integration. When addressing these challenges, we sought to adhere to the basic principals of the Agile manifesto as much as possible: making the delivery of working software our overriding goal and fostering communication and collaboration instead of dictating rigid processes. We sought to empower individuals and teams to make decisions and formulate their own processes, and to take calculated risks without fear of reprisals.

We expected that the teams would react positively to being empowered in this way and overall this assumption was correct. Most of the Scrum teams rapidly evolved to highly performing teams and demonstrated tremendous productivity.

However, we found that some of the qualities and practices that made the teams work effectively in isolation actually reduced their effectiveness in the large-scale environment.

For example: in smaller projects, we work aggressively to avoid external distractions and encourage team members to focus exclusively on their individual team goals as expressed in the backlog. In large-scale projects, team members must participate in cross-team activities and allocate time to project-wide activities. In particular, team technical leads were required to dedicate significant portions of their time outside of the team room. We found that many of our team leads struggled with this task. While they recognized the need for cross-team communication, they resented context switching and the impact on their time dedicated to the team's sprint goals.

Furthermore, all of the teams struggled with balancing their identity as a team member against their membership in the larger project. This was reflected most keenly in our failure to sustain continuous integration and produce frequent green builds. Each of

the Core Application teams were capable of frequently producing green builds in isolation, but found the additional effort and focus required to resolve cross-module issues challenging and time-consuming. While any Agile team must make trade-offs between stabilizing an application and completing more story cards, the size of this project required teams to put significantly more time into tracking down issues that were caused by other teams or dependencies. Many team members found this frustrating and demoralizing.

On the one hand, we felt that we were successful in finding effective Agile team members; a talented group of developers, product analysts, and testers who could operate effectively in a small team setting. However, we were less successful in finding a sufficient number of people who had the needed behaviors to scale the project to the desired size.

So what are the characteristics of an effective large-scale Agilist? Beyond successfully adopting the principals and practices of Agile software development, these team members must possess the ability to think and act like Global Citizens or "Tribe Members."

When the team is part of a larger community, an individual needs to look at other people's problems and sometimes sacrifice team efficiency for the sake of the program. At times, these behaviors may seem counter to Agile principles.

From our experience, members of high performing teams within a large-scale agile project exhibited the following behaviors:

- Think and operate outside team room walls. These individuals did not allow an "us versus them" mentality to exist between teams. They wanted to move around, work with different people, change focus, and take on different responsibilities through the life of the project.
- Think like a Business Owner, not just a Product Owner. On a single team, the business and the team are easily aligned, but with multiple teams and product groups, individuals need to think about what the best overall result would be, even if that means a team is sacrificed. These individuals think about the end result, not just finishing what is on the team's backlog.
- Raise issues and fight aggressively for the program's attention to them. In a small team, it is much easier to communicate your concerns at stand-up and by sitting in the same team area, but in a large setting individuals need to communicate to a broader audience and fight for priority against a host of competing issues. These individuals have to be comfortable communicating uncomfortable truths, ready to debate issues in unfriendly forums, and persistent enough to drive issues to conclusion.
- Accept change and trust decisions made out of one's sphere of control. On a small project it is relatively simple to fully communicate the drivers for a decision to all team members and come to a consensus. On projects of this scale, ensuring that everyone fully buys into a decision or change is tremendously difficult. Solving problems on such a scale will force changes upon a team that an individual might not want to adopt (e.g. employing specific testing methods to support another group's requirements or use of a specific toolset to ensure commonality). The trust that team members have for one another must be extended across the scope of the project.

Without critical mass of individuals with these behaviors on every Scrum team, it was challenging to meet our goals. In our experience, these behaviors were difficult to find as they ran counter to behaviors that have been historically rewarded in the industry (e.g., individuals working independently all night to complete a project). Ultimately, we scaled back the number of teams and successfully delivered.

6. Conclusion

When considering a large-scale development effort, ensuring that you have a sufficient number of people with the desired behaviors is just as important as having the appropriate infrastructure and processes. So the challenge is locating these people. It can be easy to find people who are interested in working on a big Agile project. It is much more difficult finding people who you want on your big Agile project. Rule #1: Don't base your decision on Agile experience! Agile is a mindset, not a skill set. Some people exhibit these behaviors intuitively, others spend years in an Agile environment and have not embraced these behaviors.

10. References

[1] Beck, Kent, *Extreme Programming Explained*, Addison-Wesley, 2000.

[2] Schwaber, Ken and Mike Beedle, *Agile Software Development with Scrum*, Prentice Hall, 2001.

[3] Leffingwell, Dean, *Scaling Software Agility: Best Practices for Large Enterprises*, Addison-Wesley Professional, 2007.