28TH ANNUAL
PACIFIC NW
SOFTWARE
QUALITY
CONFERENCE

OCTOBER 18TH – 19TH, 2010

ACHIEVING QUALITY IN A COMPLEX ENVIRONMENT

Conference Paper Excerpt from the

CONFERENCE PROCEEDINGS

Scaling Agile Practices; Replicating the small team achievements to larger projects

Don Hanson
Don_hanson@mcafee.com

Abstract

Have you ever played the telephone game where a sentence is whispered to the next person in line? It's amazing how much the sentence changes as more people are added to the line.

This game illustrates how working with larger groups of people introduces new challenges. Communication issues in this instance. Likewise larger software development projects can introduce new challenges for agile teams to overcome.

In this paper we'll look at different techniques for addressing three challenges commonly associated with larger projects:

- **Scaling the development team size** - It's easy to know what everyone is doing without taking up most of your day when there are three or four of you. This is much harder to do without spending more time as the development team grows to 10, 20 or more.
- **Big projects have big features** - You're much more likely to run into features which will require multiple iterations to complete on larger projects. How do you handle these iteration spanning features in an agile manner?
- **Reducing the post development end game** - External factors can often increase the time between completion of feature development and the software available for sale.

These approaches may help address similar issues on your projects.

Biography

Don Hanson founded his first company and began writing commercial software in the early 90's developing an evolving line of animation plug-in products. He has since worked on projects ranging from the mobile client for a wireless navigation startup to enterprise-level commercial software products. Don is the development manager for McAfee's flagship enterprise security management product.

Scaling Agile Practices; replicating the small team achievements to larger projects

Have you ever played the telephone game where a sentence is whispered to the next person in line? It's amazing how much the sentence changes as more people are added to the line.

This game illustrates how working with larger groups of people introduces new challenges. Communication issues in this instance. Likewise larger software development projects can introduce new challenges for agile teams to overcome.

In this paper we'll look at different techniques for addressing three challenges commonly associated with larger projects:

- **Scaling the development team size** - It's easy to know what everyone is doing without taking up most of your day when there are three or four of you. This is much harder to do without spending more time as the development team grows to 10, 20 or more.
- **Big projects have big features** - You're almost guaranteed to run into features which will require multiple iterations to complete on larger projects. How do you handle these iteration spanning features in an agile manner?
- **Reducing the post development end game** - External factors can often increase the time between completion of feature development and the software available for sale.

These approaches may help address similar issues on your projects.

1. SCALING THE DEVELOPMENT TEAM SIZE

It's easy to know what everyone is doing when there are three or four of you. It gets much harder as the dev team grows to 6, 7, 10 or more. An approach we've had success with is to focus on scaling team organization and communication. How we address the former influences our options for the latter.

Communication Overload
The style of communication and level of control possible on a 3~4 person team is dramatically different from what is possible with 10 or more people. Consider the different experience you have going to a restaurant with three friends compared to a party of 10 people.

Ever notice how larger parties start off pretty quiet, not much discussion going on? Then someone starts up a conversation and most people listen in, information distribution. Eventually, if you're lucky, it breaks up into two or three smaller conversations.

Conversations
The key to effective high-bandwidth communication on larger teams is to scale the number of conversations across the team while maintaining the number of people in any given conversation and the number of conversations each team member must participate in to do their job. This subtle but important distinction is a common pitfall for many teams.

If not recognized the early stages of team growth can result in a conflicting and very negative experience for team members. On a small team you're often more successful dealing with the statement of "There's no time to do my work" as a sign of resistance to high bandwidth communication. As the team grows we may hear the same statement for very different reasons. Our previous success can blind us to the reality that we're facing a very different situation.

We want to keep small groups of people focused on the same thing. We want that high band-width communication going on. We want people focusing and talking. For example, with 10 developers we may want 3~4 conversations going on the same time.

Feature teams
We know what we want but how do we get there? How do we empower small groups to have those conversations? We want each team member to spend most of their discussion time each week on features they are working on. If they are working on a feature then it should imply they have some level of control over it. The concept of feature teams are a natural fit.

A feature team is typically a cross functional group tasked with delivering shippable features and workflows. This differs from the traditional pattern where teams are organized by component or layer, e.g. one person handles the UI, another focuses on the database and so on.

Organizing the larger team into small groups of people with all the skills needed to deliver their feature combined with keeping the focus on delivering working features helps ensure each conversation has the right set of people involved and each individual is not overloaded with conversations that don't necessarily pertain to them.

Good Organization Is Critical
I come from a military family. The Table of Organization and Equipment (TOE), aka "org chart", was a concept we were familiar with at an early age. There's more to creating an effective organization of a large team than simply making it look like a pyramid.

Questions specific to your situation and product must be considered, such as how much oversight is needed? Are you making medical equipment or video games? Does your team work relatively alone or in tight coordination with others to deliver your product?

Currently our product engineering team consists of 10 distinct development teams that work together to deliver a single release; co-located server foundation, server application and client teams, and remote client team, each with Dev and QA teams, and design and tech pub teams. Some of these teams have doubled or tripled in size in recent years.

We've integrated a simple chain of command into our development process to enable the appropriate amount of high bandwidth communication and rapid decision making within the large server application Dev & QA teams, and between the other teams. There are three levels involved in the daily creation of software; feature teams, leads, managers & architects.

- Feature teams are comprised of one or more developers and QA, a Tech Pubs contact and a Design contact. Each feature team has a Dev and QA lead to represent the status of the feature area to others and be the point of contact for questions.
- Next up are the Dev/QA team and technical leads. This is a thin layer that handles immediate coordination, such as balancing user stories among team members within an iteration, and technical oversight of key areas, such as database performance. These guys are your master sergeants who keep the wheels turning.
- Finally we have the managers and architects, who share many of the same goals, have over-lapping interests and distinct but complimentary areas of authority. They tell the team what hill to take and the approach the team will use.

Not exactly rocket science so far. What has made it work well for us has been the focus on enabling decision making throughout the chain with appropriate review. We want higher levels to set broader directions while enabling more detailed design and implementation at lower levels. We want to avoid embarrassing I-wish-someone-had-told-me-it-worked-that-way mistakes while minimizing the time spent in reviews and meetings. And finally we want the team to be able to move forward while people are on vacation.

Some things we change during an iteration retrospective, for others we may wait for the next project. Every situation is different. For example, here are the current owners for some of the Dev team artifacts.

- Themes - owned by managers (who have more schedule visibility across the ten teams)
- Designs - owned by feature teams
- Stories & tasks - owned by implementers

Keeping In Sync
During the re-plan phase before an iteration we invest in two team meetings to help communicate project status in a meaningful way.

During the iteration demo the feature teams show off the new working features and workflows on running code on a demo server located in our intranet. The meeting is webcast and by request saved for 30 days for later viewing.

After watching a small but steady stream of wow-I-wish-we'd-thought-of-that-earlier suggestions raised by the wider project team during the iteration demos, we added an iteration preview working lunch. We order in some pizza for lunch and feature teams describe verbally or using storyboard/prototype what they will be creating next iteration

2. BIG PROJECTS HAVE BIG FEATURES

How to build those BIG features in an agile manner? A common pitfall for all Agile projects is losing the focus on iterative development and taking on an incremental approach. This can be hard to avoid on smaller projects but seems to go hand in hand on larger efforts.

With so much to do on a large project it's easy to fall into the trap of focusing on delivering "complete" chunks of functionality. Growing up we've learned to judge our progress on a particular task by how much we've completed. As we attempt larger and more complex projects outside our comfort zone it's hard to resist falling back on old habits.

It is worth noting that the path to incremental development is a slippery slope. At the project level iterative and incremental are not mutually exclusive. It is quite possible to use an iterative approach delivering one major feature and an incremental approach on another.

Incremental VS Iterative
The problem with delivering shippable chunks of functionality is that it requires the end goal to be well defined from the start of the project. One of the strengths of Agile development is that it allows the details of the end solution to be defined over the course of the project.

For example, suppose a project starts with the intent to build online community software with message forums but midway through the project it is discovered that more of a focus on a question and answer system is needed.

At this point with an incremental approach very few of the workflows are working so the impact to the overall user experience is unclear. In addition, a good deal of effort will have been wasted on "completed" components that now need to be changed or discarded.

However with an iterative approach, where most of the workflows are functional (if unrefined), understanding the user experience impact should be quite possible. The level of rework should be fairly minimal as impacted features are still being refined.

Jeff Patton from Thoughtworks has an excellent presentation on the differences between incremental and iterative development, along with strategies for dealing with external stakeholder's fear of uncertainty on what will ultimately be delivered. It is well worth the read and I highly recommend checking it out.

The focus here is on helping the development team maintain an iterative approach. Jeff provides an easy check to see if your project is iterative or incremental; "it's not iterative if you do it only once"[1].

Try a New Perspective
For a variety of reasons it can sometimes be challenging for an agile team to reorganize the incremental delivery of a feature along iterative lines. There are usually very reasonable sounding rationalizations as to why this particular feature is "different".

We've found that looking at it from the viewpoint of other stakeholder can help. A fresh perspective can make all the difference. Now where are we going to find other stakeholders that have the time, inclination and familiarity to offer advice? Feature teams to the rescue!

Fortunately delivering functionality iteratively does get easier with experience.

3. REDUCING THE POST DEVELOPMENT END GAME

Do months go by after feature development stops, aka feature complete, before customers can purchase your software?

Upper Management Buy In
The most important part of this process is getting buy in from upper management that changing the situation is a priority. It's amazing what can be achieved when everyone shares a common goal. This critical piece of support changes the problem from "how are we going to do this?" to "what's stopping you from doing this?"

Break Large Problem into Smaller Ones
To address this seemingly impossible problem we used the simple technique of breaking larger problems into smaller ones. Instead of trying to solve everything at once we separated out approach into three stages;

1. Define the specific problems faced by each stakeholder
2. Define a set of measurable, high level goals that would address those problems to the stakeholder's satisfaction
3. Create a detailed, measureable plan to achieve those goals

Big Systems Have Many Customers
Each stakeholder was first asked to identify the challenges preventing them from committing to reducing the end game by half, from six months down to three. In our case, a long compatibility testing cycle with 50+ products that integrate into our enterprise security management console was cited as one of the main reasons for our long end game.

Bring Quality Forward
Reducing the compatibility testing was considered too risky. Starting the testing earlier seemed the only viable option. To make this happen we needed to bring quality forward so the compatibility testing could begin prior to reaching Feature Complete (FC). This was selected as a key high level goal for Dev, QA, Design and Tech Pubs.

It's All in the Details
In a series of meeting the details of what the high level goal of reaching FC with higher quality means to each team was defined. Plans for achieving and measuring our progress towards these detailed goals were worked out stakeholder.

[1] Embrace Uncertainty: Strategies for on-time delivery in an uncertain world. 2008. <http://www.agileproductdesign.com/presentations/index.html>

In addition to the many specific measurable goals, certain process tweaks were included, such as forming the feature teams earlier to increase the calendar time available for the conceptual and high level design work. For Tech Pubs this means writing the conceptual descriptions for features much earlier in the project and detailing the workflows during a second pass later in the project.

With that we were on our way. Is this large process change worth doing? Absolutely! Is there risk associated with the change? Yes. (Or we would have made the change a long time ago) Do we have fallback plans for this large project risk like a good development project leadership team should? You bet we do!

From Day One

We were pleasantly surprised at the number of the end game processes are under our control. We went through the exercise of listing out all of the high level end game tasks and asking if they could be done "from day one" of the next project.

For example, upgrade and migration issues caused a number of headaches in the last release. However, migration is under the team's control. For the next release we will have automated upgrade testing working from day one! A number of you already spotted this and you're right; "day one" might mean week one or first iteration code drop. But the concept holds true.

If you think about it working upgrade from day one is probably one of the easiest stories in all of the first iteration. What does upgrade consist of when nothing has had time to change? It could be as little as the install wizard runs without graphics and the version gets updated.

With this approach we've turned a painful integration task into an early warning radar from day one.