Success Patterns for Distributed Agile Development

WATCHIT OPENING

WatchIT programs feature leading experts who provide comprehensive understanding and perspectives on technology trends, issues, and their business value.

Knowledge, advice, best practices: you get all this and more from WatchIT.

Program Introduction, Dr. Jim Walsh Phd.

WALSH: Agile based methodologies are a top choice for software development because of its results orientation, and ability to accommodate change. Modern development teams, however, often find themselves distributed across locations and time zones. Many Agile gurus claim that you can't do Agile successfully if you team is distributed, so what we'd like to talk about in this presentation today is how to get Agile benefits with your distributed team.

AGENDA

Today, we're going to talk about Scrum very briefly, which is a topic in itself. We're going to talk about why organizations struggle with distributed Agile. We're going to talk about various patterns of work distribution that can work in certain circumstances and have drawbacks in other circumstances. We'll discuss briefly the governance of distributed teams and finally we'll wrap up with some conclusions and tips.

LEARNING OBJECTIVES

Our learning objectives for this presentation are so that you'll know where organizations go wrong with distributed Agile, which patterns of work distribution to use in each particular situation, and the pitfalls and challenges for each of the patterns we'll be discussing.

Dr. Jim Walsh, CTO of Global Logic Inc.

First I would like to introduce myself. My name is Jim Walsh. I'm Chief Technology Officer of GlobalLogic. I have my PhD in Physics and I've been working in software product development since 1982 with a wide variety of methodologies which have been in fashion during that time. Some of the companies I've worked with have been Borland, Rational, NeXT and Apple.

I began using Agile extreme programming in 2001, when I was VP of product development at a Silicon Valley startup, and then I began introducing distributed Agile into a larger organization, and later began using Scrum in 2004.

Now let me introduce the company I work for and tell you why that's relevant. GlobalLogic is a 5,000 person software R&D services company with engineers distributed across the world. Our primary development centers are in Ukraine, Argentina, and India with smaller development centers in China, U.S. and other geographies. The reason this is relevant is that all of the projects we do are distributed, hundreds of them at any given time, so we get the opportunity to see what works and what doesn't on a regular basis with existing customers and with new customers.

Despite the challenges, Scrum is our preferred methodology, and we approach it from the point of view of practitioners, the people who actually have to deliver results on a regular basis, just like you do. We're not pundits or theorists of Scrum, but we're people who use it everyday to actually deliver results.

Additional Resources

If you're viewing this program via the Internet or on CD, you'll have access to the program transcript, a glossary of terms, key documents, links to related websites, and recommended books and articles.

A Definition of Scrum

First, I'd like to talk a little bit about the definition of Scrum and Agile, and you'll probably note in this presentation I have the tendency to use the terms interchangeably, which they are not. I want to at least start out by defining them. Agile is a set of principles for software development. You can find the list in agilemanifesto.org and the list of signatories as well.

The principles include things like people over process and working software over planning. These principles, any software methodology which embodies these principles, can be considered Agile, though Agile has come to represent a set of practices as well, for example test driven development, continuous integration and so on, which I'll freely intermingle.

Scrum is a project management framework which is well suited to Agile, which we'll discuss in more detail. Our goal here at GlobalLogic is to have a pragmatic application of Scrum to distributed development. We're not interested in replacing Scrum with another methodology. We just want to apply it in full rigor to our particular situation, which is distributed product development.

Scrum Refresher

I'll give you a very, very quick refresher of Scrum. This is an entire topic or series in itself, but just as a rapid orientation for those of you who may be a little bit rusty. Scrum has three important roles. There's a product owner, which is the person with ultimate responsibility for success of the product in the business sense and who has the main knowledge. You could think of this person in the typical software company as a product marketing person whose responsibility is to get the right product to the market at the right time.

Everybody who actually works on developing the product is considered a member of the team. Everybody on the team is peers. There's no management structure within a team, however, one of the people in the team is designated to have the role of Scrum Master. The Scrum Master serves as the external representative of the team, and also as the person who helps the team overcome obstacles and coordinates activities with other Scrum teams, but the Scrum team is an autonomous egalitarian unit.

The people working on the Scrum team deliver products in short intervals of time of typically one to three weeks which are called sprints. The objective at the end of a sprint is to have a working product which you could theoretically deploy. I say theoretically because in some cases you can actually deploy the product; some people use Scrum in that way. In other cases, there would be a short productization period at the end of the last sprint to bring the product to full shippable condition. That might involve packaging it or somewhat.

Also part of Scrum is on a daily basis, progress is made, either by continuous build and integration or some other mechanism which shows the daily output of the Scrum team. The other thing that is key to note about Scrum is that products are developed in increments of functional units. Each sprint delivers business value to an end user. It's not a technical thing, something like design, database, interface, schema, would not be a deliverable unit within Scrum. It would be something much more business oriented.

WHY ORGANIZATIONS STRUGGLE WITH DISTRIBUTED AGILE

In this section, we'll explore why organizations struggle with distributed Scrum. We see how doing it wrong can lead to lots of late night/early morning meetings, stress, finger pointing, us versus them thinking, attrition and productivity bottlenecks, which are exactly the opposite of what you'd like to accomplish with the Scrum methodology.

Scrum Failure Points

Some of the failure points that we see working with organizations that are beginning to adopt Scrum, or who think they already have adopted Scrum, is that they're not actually doing Scrum in the first place. We'll talk more about that in detail in a few minutes.

We also see engineers thrust into management roles that they may or may not be suited for. We also see that they don't have adequate tools or governance mechanisms

in place to make asynchronous or synchronous communications successful that are required for the success of these patterns. Also, we sometimes see an us versus them mindset develop between headquarters and branch offices doing distributed development or vendor versus customers, and also, probably most important is the unbalanced distribution of work and ownership, which can lead to de-motivation of the team.

Not Doing Basic Scrum

Let's talk about not doing basic Scrum. This is an entire presentation in itself, so I'm only going to cover this very lightly here. Some organizations we find fault distributed Scrum, say it's not working, when they're not actually doing Scrum in the first place; that their faulty basic implementation of Scrum would cause problems even if everyone were seated in the same room around the same table, so it's not the fault of distribution.

Also, we find that many teams who say or even think, on basis of evidence, that they're doing Agile or Scrum, are not. We'll talk more about that in a little bit. Most importantly and straight out of a Dilbert cartoon perhaps, but also occurring in real life, is a lack of process and documentation is not the same thing as doing Agile. Agile, when done correctly, is an extremely disciplined process.

Let's go a little bit deeper into some of the ways that people can fail using basic Scrum, some of the indicators that you can concretely observe. Some of the things that we ask about when we're trying to assess whether an organization is actually using Scrum or not is how long are your sprints? If they're four weeks or longer, they're probably not really doing Scrum. They're doing something like an iterative waterfall methodology. If they don't have a concrete objective definition of Done, which is applied to all tasks and user stories at the end of every sprint, then they're probably not doing Scrum.

Remember, I talked about how each sprint yields a potentially shippable unit of business value which is valuable to end users. If you say how do you know you're done and they can't tell you, they're probably not doing Scrum. If sprints are frequently interrupted, and tasks and priorities are changed mid-sprint, this is an indicator that they're not doing good Scrum. If they don't have a full build system integration and deployment cycle in place that's run at least once a day and ideally for every check in, that's also a tip off. Or task estimates not being done by the people who actually do the work.

Scrum is a very egalitarian methodology and the people who do the work actually do the estimations, the committed estimates for implementation, which actually I find is the best way to do it, even in a waterfall methodology, but in Scrum, it's one of the basic principles.

Typical problems you would see, when a team says they're doing Scrum and is actually not, is late releases, last minute milestone slips and delays, post-release quality issues, a surprise failure to meet requirements, surprise meaning at the end of a release rather

than at the end of a sprint, a lack of project predictability in general. These are all indicators of non-Scrum behavior because these are the very problems that Scrum and Agile address and that with a healthy Agile implementation there should be no surprises and that all of the surprises are dealt with as they come along. There's no tail end loading or deferral of risks. They're dealt with in a linear fashion throughout the release.

Distributed Scrum Issues

Some distributed Scrum problem indicators, we talked about basic Scrum problem indicators a second ago; now we're talking about distributed Scrum problem indicators, are: lots of late night and early morning meetings and eventual burnout and attrition within your team, either your local team, your remote team, your vendor's team, or whatever. Bottlenecks waiting for engineer to engineer round trips where productivity seems to be suffering because those people over in the remote location haven't gotten back to me yet, wherever the remote location is relative to those people.

There could be a high degree of stress and finger pointing and us versus them language or thinking. These are all tip offs that there is some problem with the way the work has been distributed.

Engineers in Management Roles

Another problem that we mentioned that can come up in distributed Scrum is; a lot of ways of doing it tend to thrust engineers into essentially management delegation roles, or supervisory roles. Engineers are not chosen and promoted based on their management and delegation abilities. They're chosen and promoted based on their technical talent. Some engineers have a natural talent for doing this, remote communication to delegation, and so on. Some do not and it's just the way engineers work, right?

Thrusting engineers in general into management and lead roles, without consideration for their aptitude for it, is costly in that it leads to a failure of the distributed model, and it also takes some of your top technical talent off-line for engineering activity, so those patterns that place engineers in management roles can be very problematic. We'll go into detail on that in a little bit.

Communication

Another problem is synchronous communication. Many group organizations have evolved expecting you'll be able to have real time, ad hoc communication with everybody on the team. This does work in the distributed pattern in general, especially when you have extreme time zones. As many of you know who've worked with distributed development, the 12 hour time difference between the U.S. West Coast and India or the U.S. East Coast and China means that there are zero working hours that the two geographies have in common, so somebody is either staying late or getting up

very early in order to talk. This can be exhausting if it's a requirement of the methodology, which it's not when done properly.

Where round trip communication is required, as we mentioned before, it can take 24 hours or even longer to get even simple questions answered, so having threaded responses across geographies severely reduces team productivity. Good tools, Wikis, forums, commercial products, help, but they don't solve the whole problem because the Earth is the shape that it is, it is the size that it is and those time zones have to be dealt with.

“Us” Versus “Them” Thinking

Another issue that comes up is us versus them thinking. That's something that seems to be very deeply rooted in the human psyche about trusting people who you see the most often and distrusting those who are far away from you. Sometimes added to that, there's a very real risk of job fears: what if this outsourced work is successful. What happens to my job onshore? Or what happens if our branch office does a better job than we do? I could be at risk here. These are similar issues that we find, whether you're dealing with a customer vendor situation, in an outsourced environment, or a headquarters branch office situation. It comes back to that basic human xenophobia of the people that you see most often are the ones that you tend to trust.

One Team Approach

All success patterns depend basically on taking a one team approach. Where you're dealing with an offshore "vendor", it's better to treat your vendor like a partner, rather than a vendor, like a temporary employee. Where you're dealing with a branch office of your own company, you have to start thinking of yourselves as a single team working in one company.

Healthy competition between teams in different geographies is good. Competing to say who's the best, who can produce the most value for the company, that's a good thing, but undermining the other team is not good. You have to keep in mind that their success is your success when you're working in a distributed team trying to deliver a single product.

Distributed Scrum Issues

Almost all issues with distributed Scrum tend to come down to the pattern of work distribution, which is where we're going to spend the bulk of our time today. The key to a successful work distribution is to align the degree of ownership that a team feels with the work that they're doing. Where those are not aligned, you head into problems. Where they are aligned, you get maximum productivity and all of the benefits of Agile.

COMMONLY SEEN PATTERNS OF DISTRIBUTED AGILE DEVELOPMENT

In this next section we're going to talk about commonly seen patterns of distributed Agile development. In this section, we're going to examine some of the common work distribution patterns we see and use with Scrum. None of these patterns are ideal in every situation and they all have positives and negatives, so we're going to be examining the challenges and advantages of each of these methods.

Local Control, Distributed Teams

The first method that we see very commonly is called local control, distributed team. In this model, the product owner, which is the stick figure on the left side of the screen under geo 1, is in its specific geography. The Scrum Master is in that same geography, and the entire team works off the same sprint backlog, in other words, the same lists of tasks to be accomplished within a given sprint.

The team itself, however, the members of the team, may be geographically distributed. We actually have worked with some companies where no two members of the team are in the same location. Everybody's working from home, distributed across multiple geographies. The distribution is both geographical and functional that the testers, developers, documentarians, UI designers may all be in different locations. Geo 1 here might indicate the company's headquarters onshore, let's say, on the East Coast of the United States. Geo 2 may be a team in Ukraine or Argentina.

We find that this pattern, which we call local control, distributed team, is most useful when the total team size is small. For example, a single four to eight person Scrum team, or a small number of Scrum teams. It works best when team members are peers, so that little or no mentorship is required. Each person is individually fully capable of doing their job without needing tutoring from somebody else, except perhaps in what's going on most recently - the most recent features.

There ideally is a natural overlap of about four to five core working hours per day in all time zones. In other words, that four or five hours of my natural day overlaps four to five hours of your natural day. They might at different times and they would be if we're in different time zones. Continuous communication channel ideally should be open between all team members during those core working hours when we're all working together. This could be instant messaging, this could be an open telephone line, teleconference line, open video conference line.

Good tool support is essential when the team is distributed to enable asynchronous communication, as well as synchronous communication. Frequent face to face contact between team members is very important to maintain the team's identity. We find getting everyone together about twice a year makes people feel like they're one team and that face to face contact is very important.

Local Control, Distributed Teams - Issues

Some of the problems to watch out for in this pattern is where you have non-peer developers or too large a team; some of the engineer as manager issues, as we discussed earlier, are likely to result because an engineer will be thrust in the role of teaching another engineer how to do his or her job and they may or may not be good at it (the engineer who's doing the tutoring).

Limited face to face communications will eventually lead the remote members of the team to start feeling disempowered or detached from the outcome. This is just a human psychological fact that we haven't found a technical way to work around. Certainly, video communication can help. You feel like you know each other face to face. Casual communication through social networking and other media can certainly help as well, but people need to see each other face to face to feel like they're really working together and are colleagues.

If there's a limited natural overlap in working hours, the team members tend to get burned out if they have to shift their schedules in an unnatural way. If there's less than two hours of natural overlap in people's time and they require shifting their hours on a regular basis, you'll probably suffer a very high attrition rate because of that.

Also, if you have a lack of good real-time communication tools plus a synchronous communication support, like good Wikis and forums and so on, if you're missing those things, the efforts between the team members will gradually become uncoordinated, leading to a loss of productivity.

Remote Functional Teams

The second pattern we'd like to discuss we call remote functional team. This is very similar to the first pattern except all of the people working in one function, let's say development, are colocated and all the people working in a second function, for example test, are also colocated, but the two functions are remote from each other. For example, all the developers might be in the East Coast of the United States, all of the testers be in India or Ukraine. The product ownership and the Scrum Master tend to be in one location with a lead in the second location.

This pattern tends to be very useful when you have a disciplined Scrum implementation with artifacts that are available globally in real time. For example, where you know which user stories and you have good tool support so that you know which user stories have been completed today, so the team working in the remote geography can pick up where the team in the initial geography left off and vice versa.

It's important to have all team members remotely attend the full sprint kick off in retrospective meetings. This can be difficult with extreme timezone differences because these meetings can be four, five, six hours sometimes, though ideally they'd be less than that, but it's important to have all the teams attend the full meetings. Otherwise, they'll lose context. It's important to get the team together frequently via remote phone

or video conference. By frequently, we mean about twice a week to have some sort of team meetings.

Also having good tool support for asynchronous collaboration work and communication such as Wikis, forums, tools that track the status of user stories, defect tracking, and so on is essential to make this pattern work as it was in the first.

Also strong functional expertise and ownership in the remote geography, that the people in geography one should not have to mentor the people in geography two about how to do the basic elements of their work. Obviously they need to collaborate about what happened today, but they shouldn't have to train them in terms of skills to make this pattern most successful.

Remote Functional Teams - Issues

Some of the problems that occur in this pattern, if those conditions that we discussed are not met, is if you do not have a disciplined Scrum process or you're missing some of the artifacts, the remote team will get out of sync with the onshore - we'll call it the geo 1 team - because they won't know what the team in the other geography is doing. They'll constantly be playing catch up and they may even be a full sprint or even two sprints offset from the onshore team. This could be fixed largely through technology and more discipline about reporting status in all geographies.

The remote team members, if they can't attend the full sprint kickoffs, retrospectives and team meetings, will be missing the full context. Sometimes people compromise on this when time zone differences are extreme by having the lead attend the full meetings, which might mean stay up all night, but this also can lead to difficulties because of the telephone tag type of communication mechanism. Also, if you do not have good tool support for asynchronous communication, you can have productivity bottlenecks waiting for that information.

Virtual Teams

The third pattern we'd like to cover, we call it virtual team. Each one of these patterns has a different organization style, as you'll see. The first two are closely related. This one is completely different. This one is where all of the implementation resources are in a remote location and all of the management resources are in a different time zone. The product owner, for example, might be onshore and the entire development, task, design, architecture team might be offshore.

We actually work with one company, for example, which has three people onshore and 80 people in Ukraine. We have another company that consists of one person onshore, who is an entrepreneur, serial entrepreneur, who manages a remote team in Ukraine and he also doesn't sleep at all at night, though that's not essential to make this pattern work. The first team sleeps. They provide strategic support.

A virtual team is most useful when your domain experts are local or onshore, for example, and your product development experts are remote. For example, let's say you have a local team with a great idea who needs to get it produced fast and cheap, so they want it to be produced offshore. In this case, a virtual team is a great way to go.

Supporting product management expertise exists, or can be built, in the remote sites. This is very important because what needs to happen to make this pattern fully successful is that you have a product owner in the onshore location and what we call a proxy product owner in the remote location, who can speak for the product owner, who is onshore, when the product owner is not available. Otherwise, you're in that pattern where the product owner onshore never gets to sleep.

It's very important that your remote team has the full range of skills and staffing required for success. For example, your offshore team would have to be able to do user interface design, architecture, implementation, testing, and perhaps competitive analysis. Essentially your entire company is offshore with the brains and the profits flowing to the onshore location.

A local team needs to be comfortable with having strategic control of the activities of the offshore team rather than tactical, task by task, control. One thing that we find through experience is almost impossible, whether you're on the customer side or on the vendor's side, is having somebody 12 time zones away try to dictate task by task what each individual on the team is going to do. This leads to burnout on both sides plus there's imponderables like a person might be absent, or late for work, or early for work, or have to leave for a childcare thing, which could not be communicated adequately in advance, so it just doesn't work.

Regular face to face visits from the product owner and the proxy to the offshore team are extremely important. The serial entrepreneur I mentioned, for example, travels to Ukraine every quarter and spends time with his team; it's him in this case, to get to know them as people, so that they have that strong working relationship and they feel like they're on the same team.

Also, strong Scrum process and process discipline is very important. Some of the problems that can occur in the virtual team setup is that the remote team ends up being understaffed or doesn't have the right skill set. This can sometimes happen because a lot of startups might be on an extremely tight budget, so they have to use the virtual team route as their maximum cost saving measure, but it's a mistake in my opinion, both on the vendor's side today and on the product development side in the past to underfund your offshore team. Going offshore is your cost savings. It's a mistake to both go offshore and underfund the development of the team. It's much better to reduce the scope of what you're going to do or lengthen the time line so that you're adequately funded.

Virtual Teams - Issues

You need the full range of skills in your remote location. Some of them may be high-end people who are relatively expensive, for example, architects who can start from a blank sheet of paper to design your system.

Another thing that can occur in a virtual team situation is if the onshore team, let's call them, tries to micromanage the activities of the offshore team. It's almost impossible to do that effectively without creating burnout and resentment. It's much better to have a proxy in the offshore location who can actually do the day to day management, who is a person you trust and who is a person who's actually worked with you, so that they understand your product direction and you're totally aligned with; you can totally trust that person to watch out for your interests in a remote location.

If you don't have adequate travel or communication with your offshore team on a regular basis, the remote team will eventually drift and become disempowered. That face to face contact is extremely important. Video conference is frequently, but physical travel I would recommend quarterly in most cases.

Finally, if you have a bad Scrum process, it's not going to work because you'll be providing interrupts to the team. They won't be maximally productive, and all of the different evils that we mentioned before will come to visit you.

Remote Self-Contained Teams

The final pattern we'd like to talk about today is what we call remote self contained teams. This pattern is very similar to the virtual pattern except that your teams can be distributed in multiple locations. For example, you would have a product owner and set of self-contained Scrum teams onshore working off their own sprint backlog, and then a proxy product owner, a person who could represent the interest of the product owner, offshore managing a set of self contained Scrum teams working offshore, each of those teams working off their own sprint backlog.

You have shared resources that meet across geographies, such as a Scrum of Scrums, an architects counsel and an experts group, which we'll talk about later, to provide the cohesion and the coordination that are required between these remote Scrum teams, but your principle points of contact are actually between the product owner and the proxies and between the Scrum Masters in the remote locations. We'll talk about governance next section.

This remote self-contained team is most useful when the coupling between the product components and interfaces in your product can be defined well enough that each team can own a piece of the product separately. By ownership, I don't mean legal ownership; I mean moral ownership, that they actually feel that they can make decisions, that they are empowered to drive the evolution of that piece of the product more or less

independently from the other components. The components of your overall architectural will always depend on each other, but if the interfaces are well defined or your product consists of multiple sub-products, this pattern can work very effectively.

It works best when you have domain and product expertise in the remote location, or you can develop it. For example, one of the very large clients that we work with is in the audio and video editing space. They have a large team in Ukraine doing product R&D and they actually hired people from the broadcast media, from television/radio, to provide domain expertise for the developers who are working in Ukraine, local Ukrainian developers. We have other people who hired doctors or nurses in the offshore environment to provide medical expertise to the remote teams.

Frequent face to face visits between team members, as I've emphasized, is very important. It also is very important to this pattern that a high trust culture exists within your company, or can be created. By high trust, we mean a tolerance for mistakes, frankly, and failures. The remote team, in order to feel empowered in ownership has to have the ability to take risks.

The good thing about Scrum is the risks tend to be very self contained. You get a demo of your product every sprint, which is, let's say, two weeks, so the remote team can't drift by more than two weeks or so - one sprint - in a healthy process, but they have to be able to trust that if they've gone in the wrong direction, you'll correct them, of course, and guide them in the right direction, but that it's okay to learn and to make mistakes. Otherwise, a distributed ownership model does not work.

Finally, you need an effective governance model in place, so that cross-team, cross-geography decisions can be made, not only with full buy-in, but with full understanding of all of the various teams involved.

Remote Self-Contained Teams - Issues

Finally, let's talk about some of the problems which can occur with remote self contained teams. If your product components are too tightly coupled, that could lead to a requirement for a lot of synchronous communication between the various distributed teams. The reason for that is, for example, if introducing a bug fix in one area or a new feature in one area is likely to break something in a totally unrelated area, then you're going to have to communicate on a real-time basis in order to coordinate those efforts. So you need to loosely coupled architecture or one that you can make loosely coupled to make this really successful.

Another problem would be a lack of domain expertise remotely, so this pattern relies for success on being able to give the remote team more or less autotonomy over its area of control. For example, it might own one subcomponent of your product. It should be able to work on that without requiring synchronous communication onshore or else this pattern is going to degrade over time.

Again, if you have a lack of face to face visits or a low trust culture in place, as we discussed before, one of the teams may feel threatened by the other team and attempt to sabotage it. So working on the degree of trust between the members of your team, the sense that they are a single team, is extremely important to the success of this and every other distributed pattern. Probably this one most importantly because it grants the highest degree of empowerment to the remote teams. This, and the virtual team model.

Finally, having a lack of an effective, inclusive governance model could lead to the remote teams feeling disempowered and we'll talk in the next section about how to avoid that through effective governance.

GOVERNANCE OF REMOTE TEAMS

In this section, we examine a Scrum governance model for distributed teams, which we found to be effective in medium to large projects.

Example: 140 Person Distributed Teams

This is, for example, an actual 140 person distributed team that we worked with in the past. This model is actually geo 1, who is the U.S. East Coast, geo 3 was the U.S. West Coast, geo 4 was continental Europe, and geo 2, at the bottom, is India. There are approximately 80 people in India and approximately 60 people at the various onshore locations leading to a distributed model.

As you can see in geo 3 and geo 4, the U.S. West Coast and the continental Europe model, we actually had used pattern number two where we had functional distribution of the teams. This was done for cost triage reason, whereas the development resources, which were expensive because they were onshore, were in the onshore location, and the test resources were in the remote location. In this case in India. In geography one, the teams were self contained with their own developers and testers, but the nature of the work allowed the developers to do a lot of their own testing because it was automated. The teams in India were all self contained as well.

There were two principal governance bodies that were used in this example. One of them was the Scrum of Scrums where the Scrum Masters would meet together twice a week at 7:00 a.m. Tuesday Pacific Time. It turns out that a 7:00 a.m. Pacific Time meeting time is most convenient for everybody worldwide. That's 8:30 p.m. in India this time of year, wintertime, or 7:30 p.m. in India during the summer and the other places it's actually during working hours. In terms of the distributed team, those are good meeting timings.

We also had a product council. The product council consisted of all of the various product owners and their proxies. This product consisted of three different sub-products and then a proxy in India, who represented the interest of the onshore product owners under an overall product owner. That team also met independently on Tuesday

mornings, and on Thursday morning the Scrum of Scrum team and the product council met jointly.

Local Geo Management Structure

In terms of management structure, there's actually no reason why you can't have a functional organization chart within a Scrum team. Even though the teams themselves consist of equals and there's no manager and structure within the Scrum teams, external to the Scrum teams it's a good idea to have leads and managers who can mentor the more junior people, especially in an offshore location, like India, where a number of the people tend to be junior, so that you can have a more senior person guiding the quality of their work output. Not telling them what to do, not assigning tasks, but helping them to be a better tester, to be a better developer and so on.

In this model, the leads in the India location and managers had members distributed over the various Scrum teams that they would mentor, that they would ensure the quality of their output. Also, the leads contributed in a very significant way in what we call an experts group.There is controversy in the Scrum community about this, by the way, but what we found is that the product owner role is often larger than any one person can do. It requires brilliant judgment in architecture, in design, in business, in pricing, in customers, and there definitely needs to be a single person in charge, a single product owner, who is clearly identifiable by the team and it has the last word in decisions. But we find that that person benefits from support from a range of experts. Those experts depend upon the project, but they may consist of people like architects, designers, and those leads that we were just discussing, so we call that the experts group.

Global Working Groups

The experts group works with the whole management team to form the office of the product owner. In addition, the members of the Scrum teams have a representative, a Scrum Master, who represents them at a Scrum of Scrums which provides coordination between the different Scrum teams. For example, the work done in team A may affect the work done by team B or vice versa, or they need to coordinate their output. That coordination is done by the Scrum Masters who meet in the Scrum of Scrum meetings that we described before.

In terms of projects managers and project management office, we find that that's best if it's independent of the working groups and rather provides oversight in reporting to the senior management. Finally, the overall manager for a large group like this or a large series of projects, is typically a VP or an SVP, who would have direct reports like director of engineering, director of testing, director of support and so on or VP in those areas for a larger company. They provide the executive oversight for the entire team.

SUCEEDING WITH DISTRIBUTED AGILE

We'll conclude our program today with some thoughts and advice on succeeding with distributed Agile.

Agile Challenges

We've learned that the desire to leverage distributed resources in an Agile working environment frequently leads to challenges. Those challenges come from the following areas in the main; engineers acting as managers, extreme time zone differences coupled with processes that rely on real time communication, the development or previous existence of an us versus them mentality or set of fears which needs to be dispelled by management and finally an unbalanced mix of responsibility and ownership.

We find that success comes from adopting the right work distribution pattern for the particular situation that you're in.

Agile Tips and Advice

Some of the tips that we'd like to leave you with are to first, make sure you're really doing Agile before you blame distributed Agile for your problems. Take a look at some of the indicators that we mentioned before and some of the other presentations in this series, and do a searching inventory and see if you're really doing Agile in the first place before you blame your distribution problem on that.

Secondly, treat everyone in every geography as one team, no matter what they do and who they work for. You should be treating testers and developers as peers. You should be treating vendor employees the same as you treat your own employees and so on if you want maximum effectiveness from a distributed Agile workforce.

Finally, we suggest you make your teams as autonomous as possible by aligning the ownership of the areas that they work on with your natural product and component boundaries and your product architecture and also by eliminating governance and management models which eliminates the need for unscheduled real time communication.

One of the benefits of the two patterns that we discussed, the virtual team and the autonomous distributed teams is that all of the communication is scheduled and there's very little peer to peer communication required. The Scrum Masters have to get up early or stay up late. The product owners have to get up early and stay up late, but the team members do not.

Conclusion

Finally, although challenging, when the success patterns we've outlined are followed in the right situations, we've actually found at GlobalLogic that Agile is the best way to do distributed development. That it's worth overcoming the obstacles to get the benefits, which include empowered and highly productive teams, immediately visible and concrete results in working on your product rather than something that's a promise six months from now. You can see it today or in two weeks.

You have sustainable and flexible cooperation between geographically distributed teams. You can make full use of distributed talent pools, whether they're globally or within the United States or within Europe. And you have the ability to embrace change, that as market conditions evolve, as technologies evolve, you can easily leverage these to your advantage within your product. Thank you for your time.