SolutionsIQ

Successful Distributed Agile Team Working Patterns

By Monica Yap

Abstract

Distributed development is transforming software development processes. In a wide variety of scenarios, development personnel are extended beyond a single building or campus. In this paper, I share successful distributed team working patterns used by myself and others on distributed Agile development projects.

Introduction

Distributed development is rapidly transforming software development processes across organizations. It consists of a wide variety of scenarios in which the personnel involved in a software development project are extended beyond the traditional setting of a single building or campus. Typical distributed situations involve companies with geographically dispersed sites: city-wide, regionally, nationally, and internationally. Successful distributed development allows teams to develop high-quality software faster, leading to improved business agility and a greater capacity to handle the pressures of globalization and competition.

But the challenges of realizing these competitive advantages are significant. Chief among them is the need to communicate accurately and unambiguously across the barriers imposed by firewalls, distance, time zones, national borders, languages, or cultures – or all of these factors. These issues are further compounded by the need to manage all dimensions of the software development lifecycle – requirements, change and assets, testing, coding, etc. – in a distributed environment.

In this paper, I will share some of the common successful distributed team working patterns I have used on distributed Agile development projects, as well as those patterns identified and documented by others. These patterns complement and strengthen each other to help teams collaborate, build trust, and work together effectively as a single team, despite any distance. The patterns I review are Boot Camp, Rotating Guru, Ambassador, High Communication Modes, Shared Community and Remote Pairing.

Distributed Development Myths

Before discussing these patterns, it is important to address some common myths regarding distributed software development in general. The case studies and experiences documented in this paper dispel these myths and demonstrate that distributed development can be highly effective.

Myth: There Is No Additional Overhead

There is a common belief that the cost of distributed development is merely the sum of the development expenses at each location. In reality, the cost must include the overhead of collaboration and integration among the distributed team members, including labor, travel, and tools. For example, in order to attain the same communication "bandwidth" that a co-located team can achieve, distributed team members will incur additional time spent in team meetings with other remote team members. To facilitate the distributed teams, a set of tools and an environment to support the tools and teams will also need to be introduced and maintained.

Myth: More Specification Needs To Be Defined Up Front

Defining more of the requirements and system specifications up front may seem less risky, but this actually creates a false sense of security for the remote teams. It encourages them to deliver to the specification, when they should be interacting with the Product Owner to truly understand the requirements, get the required feedback, and incorporate it into the next iteration. So less specification should be defined up front, coupled with the implementation of quicker inspect and adapt cycles.

Myth: You Can't Do Scrum And/Or Agile With Distributed Teams

In Scrum, it is preferable for team members to be together because it provides the greatest degree of collaboration and promotes the highest level of effectiveness in working toward the same product. However, there can be a high level of effectiveness within a distributed environment. (See case studies in References.)

Myth: Distributed Development Will Generate Lower Quality

In both of the case studies, the overall defect count of distributed Agile teams is very low. Both of these studies employed Extreme Programming practices, such as continuous integration and test driven development, to ensure that the high quality level was maintained.

Can Distributed Development Be Successful?

There are actually many case studies that detail how distributed development projects using the Scrum framework and/or Extreme Programming practices can be successful. A distributed development project with Scrum and XP has a much better chance of success than projects that utilize other development processes, such as waterfall. Here are some reasons why:

The Scrum framework maintains a customer focus and affirms that all remote teams will deliver the highest-priority features first.

- Daily stand-ups or hand-offs between remote teams bridge communications and establish a framework for frequent synchronization and collaboration.
- Sprint reviews ensure that the delivered features are reviewed and that feedback is shared.

XP practices avoid integration and quality problems.

- Continuous integration creates frequent integration points between all remote teams, facilitating problem resolution as issues arise.
- Test driven development supports the simplest design with good test coverage, and assures that no additional features are created which were not requested by the Product Owner.
- Refactoring strengthens evolutionary design and helps teams resolve architectural issues.
- Pairing provides continuous peer review and keeps team members doing the least possible work.

Case Study: WDS Global

I participated in this successful project for over four years. It consisted of one global platform with XP teams in three locations – the U.S., the U.K., and Singapore. The teams used a core set of Extreme Programming practices – test driven development, pairing, refactoring, and continuous integration, in combination with the other distributed patterns described in this paper. They made use of a single integrated global code base (no branching), and a single continuous integration server. There were daily hand-offs between each location. Even though the teams operated on two-week iterations, weekly production releases were carried out with collaboration between Development and IT.

Case Study - Sirsidynix and Xebia

At the Agile 2008 conference, Jeff Sutherland, co-creator of Scrum, and Guido Schoonheim, CTO of Xebia, presented an actual case of reaching successful hyper-productivity with a distributed team using XP and Scrum. Both of these projects were large-scale (over one million lines of code), and utilized the Scrum framework and XP practices (pairing, refactoring, and continuous integration). They also employed most of the distributed patterns detailed in this paper. Here is a matrix that compares the productivity of waterfall, co-located Scrum, and distributed Scrum teams, all based on function points (FP) per developer per month:

	Collocated Scrum	Waterfall	SirsiDynix Distributed Scrum	Xebia Distributed Scrum
Person Months	54	540	827	125
Lines of Java	51,000	58,000	671,688	100,000
Function Points	959	900	12,673	1,887
FP per dev per month	17.8	1.7	15.3	15.1

Challenges That Must Be Overcome

Building Trust

Nothing is more important than this crucial element of a hyper-productive team, and building trust is essential in the formation of cohesiveness between team members. Gaining trust is a challenge when team members are distributed across different time zones, cultures, and environments, and when they also face communication, language, technical alignment, and project management issues.

When a team doesn't possess a minimum level of trust, it's more difficult to deal with challenges when they appear – it is often easy to blame and criticize the 'other' groups and the team can break down into competing tribes. When trust is strong, team members are able to work through the most difficult issues and they often create innovative solutions.

Time Zone, Language, And Cultural Barriers

It is vital to create overlapping office hours so that remote teams can communicate, resolve problems, and bridge time zones for other mission-critical tasks. More overlapping time during the day fosters better engagement and collaboration.

When team members come from different cultural backgrounds, language and cultural differences can easily create misunderstanding and generate mistrust among team members. In addition, team members in different regions generally have varying degrees of skill and technology expertise, which can create a class system between the different remote teams and hinder collaboration.

Nothing is more important for a hyper-productive tram than team.

Communication

As outlined in the section above, different time zones create communication challenges for teams. Overlapping office hours facilitate periods for discussions, problem solving, remote pairing, and other activities that contribute to a project's success. It is crucial to create as

much overlapping time during the day as possible. Team members at different locations often fall back to using low bandwidth communication channels, such as emails or documents, which generates large amounts of lost or misunderstood information. Therefore, high bandwidth communication tools such as video conferencing or desktop sharing should be used as frequently as possible.

Technical Alignments

Team members from different backgrounds and regions may have divergent preferences about technologies and tools. For example, team members in Redmond, Washington may have a bias towards Microsoft technologies, while members in China may prefer open-source technologies.

Misalignment in engineering best practices can also create conflicts between the remote teams – for example, determining whether to use test first vs. test later development practices or when to do refactoring. Other typical misalignments are coding standards, tooling, and architectural design.

Within a co-located team, these misalignments on values are generally resolved over time by day-to-day discussion and communication between team members, with team members gradually building mutual understanding.

Project And Process Management

In a co-located team, the need for online project and process management is minimal—most co-located teams prefer a physical task board, and BVCs (big visible charts) in the team area to help them stay current on progress and information-sharing with other members. In a distributed environment, transparency and visibility are essential for all remote teams. High-visibility, real-time online project tracking and process management (for example, next Sprint planning dates, CI monitoring) enables all teams to be fully engaged in the development process.

Distributed Scrum Models

Isolated Scrums

Teams are isolated across different locations. Some teams may not be using Scrum.

Distributed Scrum Of Scrums

Cross-functional Scrum teams are isolated across locations and integrated by a Scrum-of-Scrums that meets regularly across locations. This model partitions work among cross-functional, isolated Scrum teams in different locations while eliminating most dependencies between teams. The Scrum Alliance recommends this model.

Totally Integrated Scrums

Scrum team members are distributed across locations. This model is not recommended for team members who are not experienced in Agile.

Distributed Team Patterns

To overcome the challenges above, a set of patterns can be employed. Most of them should be implemented in order to achieve a certain level of benefits and to mitigate risks.

Boot Camp

Boot camp brings all remote team members together – it can be a kick-off for a co-located project or the occasion to mark key milestones such as each release start. If the entire team cannot be brought together, a minimum crucial number of members from each remote location should come together and participate in the initial Sprints. Boot camp ensures that the team begins work on the project with a shared understanding of customer context, as well as common alignment to tooling, initial architecture design, definition of done, and code standards. Natural roles in this team will start to form – for example, "John is the CI expert, since he has the most knowledge around CI best practices and management." Most importantly, the values from the different remote team members are shared and this helps to begin building trust among them.

Rotating Guru

Even when all team members begin the project together, the understanding and

context will diminish over time, which diminishes the level of trust. Regular visits to each location by a 'rotating guru' facilitates collaboration by infusing each distributed team with the context from his home team, while simultaneously gaining context from each location – this creates a cross-pollinating effect. During the visit, the guru works as a regular team member at this location, pairing with local members as much as possible to learn how they work and to observe their development style; this fosters communication and understanding, and also creates personal trust between the guru and the local team. We found that the duration of the visit must be three weeks or more in order to be effective. A different team member should act as the rotating guru in each rotation, to help minimize the traveling burden on any one individual and to maximize the cross-pollination across all remote teams.

High Communication Modes

There is a wealth of evidence demonstrating that face-to-face communication is the most effective means of exchanging information. This occurs daily within a co-located team, but it is more challenging in a distributed environment and every effort needs to be made to achieve the same high level of engagement.

When each remote team is in a different time zone, overlapping working hours need to be established. When this is not possible, a special program needs to be created that facilitates this and offers special incentives for members to participate – for example, extra support for colleagues who choose to assume earlier or later work hours. Sometimes this cannot be easily accomplished in the office, so providing special access (VPN) or tools (Web cam, Skype phones) that enable members to work from home will facilitate adopting the new overlapping schedule.

To maintain constant high-bandwidth communication, a comprehensive suite of video conferencing, desktop sharing, and instant messaging tools should be provided – such as Web cams, Skype, VNC, Google Talk, etc. These channels must be readily available and easily accessible to all team members. For example, if the Web cam and Skype phone are in a room down the hall from the team area and must be reconfigured for each use, it is very likely they will be ignored and go unused. It is best to have a dedicated, permanent webcam with video conferencing equipment set up and ready to use in the team area, especially during the overlapping hours.

As much as possible, always involve all locations in key events during the Sprint, such

as Sprint planning, review, retrospectives, and daily stand-ups. Each team may try to form a local meeting of their own – to avoid this, try using techniques such as a 'talking stick' to ensure only one voice at any given time.

Sprint planning: Break Sprint planning into two parts. In Part I, have a joint planning session to determine what each team is working on at a high level – this helps establish a shared understanding of each other's Sprint backlog and identifies dependencies up front. In Part II, have a separate planning meeting to clarify details and break stories down into tasks. When it is impossible for all remote teams to participate due to time zone issues, for example, the Product Owner will then need to attend the Sprint planning with each team remotely. If this creates a conflict for the Product Owner, a Product Owner proxy can be appointed.

Sprint review: Even if it is impossible for all remote teams to have a joint Sprint review, it is important for each remote team to understand what others teams delivered during the Sprint. It is also important to designate a remote proxy for the Sprint review to demonstrate what the local team has accomplished.

Daily stand-ups: Depending on the size of the remote teams, it may not be practical to have all remote team members participate in each other's stand-ups. We found that it is effective to have a Scrum-of-Scrums each day to share enough information and visibility into the work being done on a daily basis. It's not only important that the ScrumMasters participate in the Scrum-of-Scrums – it's best to rotate one team member per week or per Sprint to participate as a representative for their local team.

Retrospectives: This is the most difficult of all meetings to facilitate when remote teams participate. An alternative is to have each local team hold their own retrospectives with representatives from the remote teams. The action items should always be shared and visible to all teams.

Remote Pairing

While pairing within a co-located team offers the best knowledge-sharing and fosters close collaboration among team members, remote pairing is even more crucial between remote team members. Close collaboration is the quickest way to build trust between two remote team members who come from different backgrounds and cultures.

Close collaboration is the quickest way to build trust between two remote team members

We found that when an emergency critical issue arose that needed to be resolved immediately – and it could not be handled solely by one team in one location, meaning that it had to span into the next time zone – we had great success when one team member paired with another remote team member for knowledge transfer. The understanding was much deeper and an immediate bond was created between the two who paired. Remote pairing then became a regular practice between teams.

Remote pairing should be done frequently at a fixed time and on a fixed schedule. This creates favorable side benefits over time, including avoiding code ownership within a local team and sharing the knowledge and experience gained from working with members from different locations. Periodic rotation of the remote pair ensures cross-pollination among members.

Ambassador

An ambassador is a local champion for remote team members, and as such always maintains a trusted relationship with them. This is a nice complement to applying the Rotating Guru pattern, where the rotating guru becomes the ambassador when he returns and the trust relationship can continue naturally. Without the Rotating Guru pattern, other methods would be necessary to bring the local champion and remote team members closely together. To help the ambassador build trust and rapport with remote team members, this method should always be used in conjunction with High Communication Modes and the Remote Pairing pattern.

Shared Community

'Shared community' is a virtual community across all remote teams, and it should include a knowledge base that serves as the single source for information across all teams. Without this pattern, detailed information is kept in multiple source locations, which creates a great deal of confusion and leads to large amounts of wasted time as team members try to discover the 'truth' and synchronize between the different source locations.

An approach similar to Facebook may also be helpful – posting team member profile information, such as a photo, personal interests, hobbies, etc., in a shared location fosters greater personal understanding among remote team members when face-to-face communication isn't possible. This community should be accessible online in real time by all locations, and a set of tools should be provided:

- Wiki and blogs: Provides a knowledge base where real-time edits are easy and visible.
- Online project management tool: Provides project progress transparency and real-time status updates.
- Shared mailing list and folder: Creating an easy way to message everyone and a common folder enables members to search for and read past messages.

Integrated Global Code Base And Single Continuous Integration Server

Single code repository: Everyone should check in to the same code base and resolve their conflicts upon checking in (i.e. use the main trunk without branching). A global policy that dictates when members can check in should be employed across all teams. To allow for check-in as frequently as possible, a high-bandwidth connection must be provided from all locations, and the source repository tool should provide efficient change management. Both of the case studies found that tools such as CVS and SVN were not efficient enough to accommodate fast and frequent check-ins across widely-distributed geographic locations, so in both cases a decision was made to switch to Perforce as their repositories.

Single continuous integration server: All teams should use the same server as their CI box, ensuring that builds and tests are run against all the changes at any given time. A global procedure should be employed across all teams for build breakages, such as mandating a rollback or calling in members who checked in to fix the build.

For both the single repository and the CI server, there should be a hot backup setup in each location in the case of power outages, server downtime, and other unforeseen failure situations. Should those circumstances occur, which they did in the case of WDS Global where there was a city-wide power outage for three days at one of their locations and no hot backup existed, some of the remote teams may not be able to function normally. This leads to cascading delays in other locations.

Technology Alignment

In an open and honest environment, a co-located team aligns its technology, tooling, and engineering best practices, often from frequent informal discussions – this perpetuates and reinforces trust and shared values. In a distributed environment, however, these discussions do not occur naturally, so it takes a deliberate effort to ensure the most critical elements are aligned across all remote teams and maintained over time. Without the proper alignment, teams lose trust and shared values and quickly form an 'us' vs. 'them' mentality.

Technology alignments: Coding standards, frameworks used, and the general promotion path of how to introduce new technology should be established across all teams, or else a great deal of effort will be needed to re-align these elements. This framework builds trust and shared values and helps to self-regulate team discussions so they don't often digress and focus on unresolved alignments.

Tool alignments: The integrated development environment (IDE) and commonly used tools such as those for database queries should be aligned and standardized, including their configurations. Without this alignment, a large amount of time may be spent troubleshooting the different configurations or tooling properties, especially over remote pairing and technical discussions.

Engineering best practices: It is important to establish a set of common best practices across all remote teams, such as TDD or refactoring. This is the most difficult alignment of all, since it contains a large element of subjectivity. ScrumMasters must facilitate and ensure buy-ins from all remote teams. The results should be published and maintained in the shared community, so it is viewed as definitive information and serves as part of the orientation for new team members.

Other Distributed Team Best Practices

Here are some other common idioms used by many successful distributed teams:

Product Owners daily meetings

When there are multiple product owners across remote teams, the daily meeting between the product owners allows the product backlog to be synchronized and facilitates quick resolution of detailed requirement questions or issues.

Hourly automated builds

Employ a minimum hourly build policy to ensure that integration occurs frequently across all remote teams.

No class system

No distinction should be made between members at different locations. When there is a class system, it severely hinders the trust relationship between teams and the 'lower' class teams will always feel they are left out or that they are being treated as second-class citizens. Sometimes a no-class system is difficult to maintain due to different average skill levels in each remote team, and fostering this environment may require some drastic changes such as hiring more senior members into the 'lower' class teams or relocating the senior members from one team to another.

Conclusion

Based on my own personal experience working with distributed teams over multiple years, as well as collaborating with others who have had similar experiences, embracing these patterns will set the stage for distributed teams to be successful.

References

1. “Fully Distributed Scrum: The Secret Sauce For Hyperproductive Outsourced Development Teams“ by Jeff Sutherland (Agile 2008). http://blog.xebia.com/wp-content/uploads/2008/08/fully-distributed-scrum-sutherlandxebia-agile2008.pdf

2. “Distributed Scrum: Agile Project Management with Outsourced Development Teams” (HICCS 2007). https://34slpa7u66f159hfp1fhl9aur1-wpengine.netdna-ssl.com/wp-content/uploads/2014/05/Distrubted-Scrum.pdf

3. “Follow the Sun: Distributed Extreme Programming Development,” by Monica Yap (Agile 2005)

4. “Extreme Programming Explained: Embrace Change” (2nd Edition), Beck, K. with Andres, C. Addison–Wesley, 2004.

5. Braithwaite, K., Joyce, T., “XP Expanded: Pattern for Distributed eXtreme Programming,” XP20.05

6. Wells, Don. http://www.extremeprogramming.org/

SolutionsIQ

Successful Distributed Agile Team Working Patterns

Want to Learn More?

Visit SolutionsIQ.com

Email: solutionsiq@accenture.com

Toll-Free: 1-800-235-4091

Direct: 1-425-451-2727

accenture | SOLUTIONS IQ

@SOLUTIONSIQ