

XP Expanded: Distributed Extreme Programming

Keith Braithwaite
keith.braithwaite@wdsglobal.com
Tim Joyce
tim.joyce@wdsglobal.com

WDS Global, Forelle House, Marshes End, Upton Road, Poole

Abstract. Colocation has come to be seen as a necessary precondition for obtaining the majority of the benefits of XP. Without colocation teams expect to struggle, to compromise and to trade off the benefits of XP *vs* the benefits of distributed development. We have found that you can stay true to the principles and not compromise the practices of XP in a distributed environment. Thus, business can realize both the benefits of distributed and of truly agile development.

Keywords: Agile, XP, Extreme Programming, Scrum, distributed, multi-site, outsourcing

1 Introduction

The small but growing literature on “Distributed Agile development” takes a largely pessimistic view. Often, writers assume that having the members of a team distributed widely in space (and/or time) would deal a fatal blow to the communication mechanisms upon which agile development relies. They then infer that, should one attempt to use an Agile method (for example, XP) in a distributed environment, those practices which embody the communication value in a colocated setting would be near-fatally compromised. Thus that XP itself would be compromised and various additional practices of a questionable nature (often forms of documentation, or of process automation) would need to be introduced. So it is believed that much of the benefit of “agile” development would be lost, and that winning much of the rest would be very challenging.

Note the subjunctive mood throughout the previous paragraph. Much of the Distributed Agile literature is speculative. The few reports of distributed agile development in practice are stories of hedging, compromise, and of profoundly mixed results. These are valuable data points, but do not address the question: what happens if a team distributed in space and time works as fully as possible in alignment with the principles of, say, XP?

Our observation is that such a team can succeed, and win for its business sponsors both the advantages of agile development and of distributed working.

2 Distributed Agile Development

Various terms, such as Distributed Agile Development and Distributed Extreme Programming are currently terms used to refer to a variety of process and practices associated with non-colocated development teams. When we need to refer to teams that are not colocated we prefer to say “cross-site”.

We distinguish these three general cases for non-colocated, Agile aligned development:

Agile Outsourcing (AO): Where an agile team is created at an appropriately low cost offshore location. Requirements are generated onshore, and communicated offshore using documents, people and tests. There may be some code sharing between the onshore and offshore team, but not shared ownership as commonly understood. This approach has a degree of popularity and has been widely discussed, notably in [10] and [14].

Agile Dispersed Development (ADD): As practiced by much of the Open Source community and some commercial companies [6]. Developers tend to be physically alone, but connected through a variety of communication channels. Practices such as frequent releases and continuous integration are employed, Pair Programming¹ and other team based activities are not (or only in a very limited form). Because of this, aspects of shared code ownership are often. In the open source case, this results in practices such as Benign Dictator, and Trusted Lieutenant.

Distributed Agile Development (DAD): Customers are distributed. One development team is distributed evenly over several sites to remain close to the customers. Rich, high density communication ensures that Agile principles and practices are not compromised, locally or globally.

2.1 Wireless Data Services Case

WDS is a global business providing various services to mobile telephone network operators and handset manufacturers. These include web based software for self-serve device management. At the end of 2003, core services were developed and deployed as APIs by a UK based team using XP. In each region a (non XP) team would use these internal APIs to deliver on locally generated requirements.

At the beginning of 2004, we brought all developers together in the UK for an XP and Java “boot camp”. This time was also used to establish a single global team. Developers were then dispersed to three sites (UK, Seattle, Singapore), and Distributed Extreme Programming (DXP) begun in April 2004. The business considers the change to be successful.

¹ We distinguish the names of practices by using **this** face.

3 How to Remain Extreme Around the World

Given that there is a business need to have developers around the world work together, how can agility be preserved?

At one level, the answer is quite simple: maintain a commitment to the value judgments that characterise the core of all agile methods, the Agile Manifesto [2]. Some writers take it that in applying agile approaches to distributed working must require sophisticated tools and complicated process models [12]. This seems at odds with the spirit of the manifesto. Others report some success on small-scale pilot projects using a much more direct approach [9]. We have shown that the direct application of XP to full-scale commercial development can be successful.

The implementation of the Agile Manifesto that we prefer is XP. As described in [1], this uses various Primary and Corollary Practices to embody Principles which manifest Values. We also apply many project management ideas adopted from Scrum [3]. Of the XP Values, we find that in the DXP case, Communication and Respect are especially emphasized. The key problems in DXP are to maintain sufficiently rich communication, and a sufficient level of respect, between colleagues separated widely in time and space.

Our experience is that these problems are soluble in a way wholly aligned with Agile principles. We consider that the defining characteristics of DXP are the use of: One Team, Balanced Sites and Distributed Standup, One Team, One Codebase.

4 Can Distributed Development be Truly Agile?

The question is rather: can successful Distributed Development be anything other than Agile?

4.1 Traditional Distributed Development

We did not investigate the literature particularly thoroughly before experimenting with DXP². Instead, we expressed the XP value of Courage. Confident that the principles of Agile development, and the practices of XP and Scrum, were fundamentally sound we started with the obvious first steps to implement DXP.

Subsequently we read Carmel [4]. The most interesting feature of this work (which predates the Agile revival) is that most of the content is aimed at convincing a plan-following, top-down, command-and-control manager of the value judgments captured in the Agile Manifesto. Carmel identifies “loss of teamness” and “loss of communication richness” as two of the five centrifugal forces that will damage a global team. His solutions revolve around such items as “lateral communication” (communication between co-workers across the width of the organization chart), encouraging a common team culture, building trust through face-to-face meetings, and so on.

² Perhaps if we had, would have been scared off.

It's our claim that almost all of the best practices presented by Carmel for building dispersed traditional development teams will be second nature to an organisation practiced at XP.

4.2 The Colocation Shibboleth

Having all team members in one room is a defining characteristic of default XP. Beck gives **Sit Together** as the first Primary Practice of XP. However, he also states clearly that “[...] teams can be distributed and do XP”.

By definition, distributed teams cannot achieve **Sit Together** as a whole, although each regional group can and should be colocated itself (as developers in WDS's regions are). However, if we look beyond the one-room practice to the value of Communication it manifests we can see the possibility of expressing that value in other ways.

We submit that the injunction to put everyone in one room is an absolutely necessary rule to apply when *introducing* XP. Non-agile development practice often trains developers to be solitary, uncommunicative and non collaborative. Together with pairing, **Sit Together** is very effective at breaking those habits—as required to roll out the rest of XP. But, if a body of developers are available already trained to work gregariously, to communicate as much as possible, to seek out collaborators, then perhaps the need for colocation as the prime mechanism to manifest the Communication value is weakened.

Beck's discussion describes the XP practices as *theories*, with attached predictions. The **Sit Together** theory predicts that “[...] the more face time you have, the more humane and productive the project.” We would generalise this to state that the more, more rich, communication you have, the more humane and productive the project. Face time is still much preferred, but it turns out not to be a *uniquely* valuable medium.

5 Practices for DXP

We have identified a number of practices for cross-site development, with particular emphasis on agile techniques. We present them here as candidate patterns in something like Portland Form [5]. The patterns are organised by thematic area, and ordered by significance within an area—as indicated by the number of *'s suffixed to the name.

5.1 People

These practices relate mostly to human interactions, the most difficult and also most crucial aspect of DXP. Each site implementing these practices needs people with experience of co-located XP.

One Team ***

Business needs lead to development resources distributed widely in space and time. Communication between members is compromised. Trust and cooperation can break down.

Therefore: Maintain as far as possible a single team identity across all locations. Encourage non-business communication, encourage any activity that lets team members share a joke or a cultural reference. Cherish every successful interaction. Let people play. A high level trust is maintained, resulting in fewer conflicts. When work related conflicts arise, a joke can defuse the tension.

Relates to: Kickoff; Multiple Communication Modes; One Team, One Codebase; One Team, One Build

As seen in: Seems to be novel as stated, we'd love to learn otherwise

Balanced Sites ***

Team members at one location are sometimes blocked while waiting for actions or decisions taken at another site. This creates resentment on both sides; the dependent site resents the productivity impact and the loss of decision making power; the depended site resents the interruption of thought and activity.

Therefore: Make all sites equal in skill and numbers, and empowered to take any decision, so that inter-site dependencies are minimized. There is no delay between when a decision or action is required and when it is performed, maintaining flow. All team members feel fully engaged.

Relates to: One Team; Distributed Standup (even though dependencies are minimized, force communication anyway); Ambassador

As seen in: Seems to be novel, as described. We'd love to learn otherwise

Ambassador **

Members of a team in one location find it hard to understand the point of view of members in another location. Trust and cooperation break down, it is hard for one local group to work effectively with another.

Therefore: Send an ambassador from one region to another, for an extended period. Such a local representative can interpret the communications of the remote group, demonstrate that "they" are just like "us", and influence locally on behalf of the remote group when required. Ambassadors also carry business domain knowledge between sites.

Relates to: Visits Build Trust; One Team; Balanced Sites; Multiple Communication Modes

As seen in: [8], [11], [7]

Visits Build Trust **

Team members find it hard to have faith in the good intentions of remote colleagues. Blamestorming replaces collaboration, fingerpointing replaces problem solving.

Therefore: Have team members rotate through locations continually. Always

have at least one team member away from their home location. Trust in a team member currently remote can be maintained, based on the experience of having worked with them colocated in the past.

Relates to: AmbassadorOne Team, Multiple Communication Modes

As seen in: [4], [11], [8]

Kickoff *

A new project is to start. All team members involved must synchronise their ideas about it.

Therefore: Bring everyone involved in the project together in one place at the same time. Future distributed working is informed by a cohesive view of the project, and secure interpersonal relationships, formed while the advantages of Sit Together were available.

Relates to: One Team; Multiple Communication Modes

As seen in: [4], [8], [11]

5.2 Communication

These patterns consider communication, the lifeblood of agile development and the greatest challenge for DXP.

Distributed Standup ***

Team members remote from one another cannot easily see each other's story board, overhear technical discussions, share in resolving issues. Remote members' idea of the state of the team fall out of sync, damaging the cohesion of the team.

Therefore: Have a video conference session running whenever possible, but at least once a day, every day for each pair of sites adjacent in time. Force an overlap if required. No member can forget that the remote members have a stake, status is shared (perhaps transitively).

Relates to: One Team; Balanced Sites

As seen in: []

Multiple Communication Modes ***

The members of a team cannot be colocated. Face-to-face communication (explicit and "overhearing") cannot be used to maintain tacit knowledge. Many different kinds of knowledge must be shared, often during sharply time-limited handover slots.

Therefore: Provide team members with as many communication media as possible. At least these: individual and conference telephone, teleconference, video conference, email, IM, wiki, VNC. Communication is fostered greatly, and many

different modes of communication can be applied in parallel. A good conversation to hear at a videoconference standup meeting would be: *site 1: We had an idea for that problem, I've just jabbered you the URL for the wiki page that discusses our example code, see what you think. Site 2: Great! Let's remote-pair on this tomorrow.*

Relates to: One Team; Wiki as Shared Location; Remote Pair Ambassador; Code is Communication

As seen in: [8], [7], [4]

Wiki as Shared Location **

Team members can meet neither at the same time nor at the same place. Communication has to be both over low bandwidth channels and asynchronous. Members do not feel members of “one team” due to disjointed communication.

Therefore: Use a wiki. A shared virtual place is created where notices may be posted, asynchronous conversations take place in a persistent form, and a feeling of community fostered.

Relates to: Multiple Communication Modes; One Team; Code is Communication is dual to this practice

As seen in: Many sources mention team wiki's, but the notion of wiki as shared virtual location is not explicit. The walls of public lavatories.

Remote Pair **

Developers that need to Pair are remote. Code changes need to be shared. A familiar shared environment needs to be available to allow pair programming between sites.

Therefore: Establish an easy-to-start environment with rich communication (video, text and sound) and a shared development tool (use VNC or similar to share an IDE). Agree a regular time when remote pairing will occur. Complex, code-level decisions and communication will occur in a familiar way.

Relates to: One Team, One Codebase; Many Communication Modes; Code is Communication

As seen in: [12], many informal mentions on newsgroups, etc.

5.3 Code

These patterns address what is perhaps the easiest aspect of DXP, the technical.

One Team, One Codebase ***

Widely separated team members need to maintain a common identity as technical problem solvers. They need to share rights and responsibilities toward each others' work, just as colocated workers do.

Therefore: Have all team members everywhere use a single, shared codebase.

Technical problems and their solutions are shared. The whole team always has a common point of reference.

Relates to: **One Team; Code is Communication**

As seen in: Seems to be novel as stated, we'd love to learn otherwise. This practice largely opposed to much of the advice given in [13].

Functional Tests Capture Requirements **

Requirements need to be transmitted from one site to another. A great deal of time and energy would be consumed to make requirements documents work as a medium.

Therefore: Use failing functional tests to express the required functionality. The requirement is expressed unambiguously.

Relates to: **Code is Communication; Tests Announce Intention**

As seen in: [8]

One Team, One Build **

All team members need to share responsibility for maintaining all code in a working state. With multiple integration machines, inevitable environment skew can hide the reasons for build failure. Arguments break out between sites as to whether a break is because of “your build machine” or “our code”, destroying shared responsibility and respect.

Therefore: Have a single build server. Set up an RSS feed so that each site can hear the build passing or failing. The build status of the global codebase is a single boolean value.

Relates to: **One Team, One Team, One Codebase**

As seen in: Seems to be novel as stated, we'd love to learn otherwise. This contradicts the advice given in [13] for large teams: §3.4 states “Each [sub]team can and should set up their own automated integration server”

Code is Communication **

Colleagues far apart cannot discuss technical issues, design ideas, requirements face-to-face. This can threaten the conceptual integrity of a code base (and team).

Therefore: Use the code base as a communications medium between sites. Converse with remote colleagues via the codebase. Express problems as failing tests in a suite outside the build, express design ideas as working code in a scratch area of the repository. *Code is written for humans to read and only incidentally for computers to execute* — attributed to Knuth. A unique, unambiguous, shared artifact exists to transmit technical ideas.

Relates to: **Tests Announce Intention; Wiki as Shared Location** is dual to this practice.

As seen in: Seems to be novel as described, we'd love to learn otherwise. Although “ask the code” is a common XP slogan, it would seem to mean something rather different.

Tests Announce Intention *

Colleagues working on the same code base cannot “overhear” that they are about to collide on the same region of the code and so coordinate their efforts. Remote teams suffer integration races.

Therefore: Use functional and/or acceptance tests to publish the intention to work in a particular area. Remote colleagues can identify, asynchronously and unambiguously, what areas of the code others are likely to be changing soon.

Relates to: Code is Communication

6 Conclusion

A need for Distributed development exists in business, a desire to remain Extreme exists in the developer community. There is no need to compromise the second to accommodate the first.

Remaining firmly aligned to Agile principles allows development teams to grow with businesses as they globalize.

References

1. Beck, K. with Andres, C.: Extreme Programming Explained: Embrace Change (2nd Edition) Addison–Wesley (2004)
2. Beck, K. *et al*: The Agile Manifesto <http://www.agilemanifesto.org/>
3. Beedle, M., Schwaber, K.: Agile Software Development with Scrum Prentice Hall (2002)
4. Carmel, E.: Global Software Teams Prentice Hall (1999)
5. Cunningham, W.: About the Portland Form <http://c2.com/ppr/about/portland.html> (as at end 2004)
6. Daniels, J., Dyson, P.: CS2 Dispersed Development OT2004 <http://www.spa2005.org/ot2004/programme.shtml> (2004)
7. Jensen, B., Zilmer, A.: Cross-Continent Development Using Scrum and XP <http://www.informatik.uni-trier.de/ley/db/conf/xpu/xp2003.html> (2003)
8. Fowler, M.: Using an Agile Software Process with Offshore Development <http://martinfowler.com/articles/agileOffshore.html> (as at April 2004)
9. Kircher, M., Jain, P., Corsaro, A., Levine, D.: Distributed eXtreme Programming
10. Martin, A., Biddle, R., Noble, J.: When XP Met Outsourcing XP 2004, LNCS 3092 (2004)
11. Poole, C. J.: Distributed Product Development using Extreme Programming XP 2004, LNCS 3092 (2004)
12. Reeves, M., Zhu, J.: Moomba–A Collaborative Environment for Supporting Distributed Extreme Programming in Global Software Development. XP 2004, LNCS 3092 (2004)
13. Rogers, R. O.: Scaling Continuous Integration XP 2004, LNCS 3092 (2004)
14. Simons, M.: Internationally Agile Informit.com (2002) <http://www.informit.com/articles/article.asp?p=25929>