Using an Agile Software Process with Offshore Development

Martin Fowler

Last Significant Update: September 2003

For the last two years ThoughtWorks has operated a lab in Bangalore India to support our software development projects in North America and Europe. Traditional approaches to offshore development are based on plan-driven methodologies, but we are very firmly in the agile camp. Here I discuss our experiences and lessons learned in doing offshore agile development. So far we've discovered that we can make it work, although the benefits are still open to debate.

One of the fundamental tenets of any agile software methodology is the importance of communication between the various people involved in software development. Furthermore agile methods put a large premium on improving communication through face-to-face communication. As the agile manifesto states "The most efficient and effective method of conveying information to and within a development team is face-to-face conversation." Extreme Programming emphasizes this with its practice of a single open development space where the team can work closely together. Cockburn's book spends a lot of time talking about the importance of physical proximity in agile methods.

Another trend that's been grabbing the software development world recently is the move to offshore development, where much of the development work is done in lower paid, ahem more cost effective countries. Offshore development seems opposed to agile development in a couple of ways. For a start it immediately goes against the notion of physical proximity, since by definition offshore developers are a long way away. Secondly most offshore organizations favor the plan-driven approach where detailed requirements or designs are sent offshore to be constructed.

So the fundamental question is whether agile techniques can be used in an offshore setting. If so how does it compare to using an plan-driven methodology (the term I'll use here for non-agile)?

The experiences I'm writing about here are based on work done over the last couple of years by ThoughtWorks. We opened an office in Bangalore India in 2001 and have done several projects which have used a Bangalore based team. We've also done some offshore development with our Melbourne office. In these projects we've committed to using as much of an agile approach as possible, since we believe that agility is an approach that's in the best interests of our customers. In this essay I'll describe some of the lessons we've learned so far.

To help provide some context, it's worth talking a little about the way in which we've setup the Bangalore office. We expected the office to be used primarily as an offshore development environment, so we recruited mostly application developers. The majority were hired directly out of college and we seasoned the mix with some more experienced developers. It was very important to us that we retained our very high hiring standards (typically we only offer jobs to about 1 in 200 applicants), and we continued the model in India. As a result we have a very talented group of developers but with a mix of experience levels. We also brought over several more experienced US developers, to mentor the newer developers on both software development and the agile/XP practices we've come to enjoy. Currently we have about forty developers in Bangalore.

The Melbourne office is a very different story. Here we set up the office expecting it to work primarily on Australian work, which is now the case. In the early days of the office we did some offshore work while we were trying to get Australian work. The team is equally talented but we tended to hire a larger proportion of experienced developers.

Lessons Learned

Use Distributed Continuous Integration to Avoid Integration Headaches

I've heard several stories about problems with integrating the work across multi-site teams. Even with a lot of care put into defining interfaces, integration problems still rear their ugly head. Often this is because it's very hard to properly specify the semantics of an interface, so even if you have all the signatures right, you still get tripped up on assumptions about what the implementation will actually do.

The earliest XP practice to really take hold at ThoughtWorks was Continuous Integration, and we've become so accustomed to it that we were determined to use it with our offshore development. So from the beginning we've put everyone on a single code base, with CruiseControl running to both build and run tests. This way everyone is kept close to the mainline, whatever their location happens to be.

Everyone has been delighted with how well this works. It's main benefit is that problems that plague other groups with integration just don't happen to us. The continuous integration and test process flushes out many integration problems very quickly, as a result they can be fixed before they become hard to find.

CruiseControl's web page allows all sites to see what's going on abroad. First thing in the morning you can check the CruiseControl web page to see what changes have been made by the other sites. It provides an easy way to catch up on what's going on at the other end of the world.

This does require good build discipline, where developers strive hard not to break the build, and fix it immediately if it becomes broken. It's generally accepted practice that if you commit changes to the mainline, you should not go home until you have received the email message from CruiseControl that says that your changes resulted in a successful build. A late night bad build is much more serious when the remote office is running off the same build.

Although world-wide Continuous Integration is resoundingly popular, we have run into some problems. Communication pipes aren't as wide and reliable as we'd like, so many source control operations can get awkward from a remote site. In general we keep the build servers in the same site as the majority of developers, but remote sites can find it takes an annoyingly long time to get a fresh update from the mainline. The longer the communication lines are, the more they are prone to anything from glitches to lines being down for a while. Having the repository accessible 24 hours makes it annoying to take it down to do backups. All of these issues would mitigated by a clustered code repository, but we haven't experimented with anything like that yet.

(Interestingly people assume that these communication problems are specifically a problem with a remote site like India - but we've found problems often occur with the infrastructure in the west too. Continuous Integration requires good connetivity, often better connectivity than people are used to.)

Have Each Site Send Ambassadors to the Other Sites

As I've said above, agile methods stress the importance of face to face human interaction. Even if everyone cannot be co-located, moving some people about clearly helps a lot. From the beginning we decided to ensure that at all times there was someone from the US team present in India to facilitate the communication. Such an ambassador already knows the US based people and thus adds his personal contacts to help everyone communicate.

We've now expanded this to several levels. We found it useful to send a US developer and a US analyst to India to communicate on both technical and requirements levels. It's also valuable to send someone from India to the US team. The plane fares soon repay themselves in improved communication.

One of the benefits of a business-oriented ambassador on the offshore team is that it helps provide business context to the offshore team. Building software off just a list of requirements misses out a lot business context - developers are told what to do without being told why it's important. The why often makes a big difference in doing the what properly.

An important part of the ambassador's job is to communicate gossip. On any project there's a lot of informal communication. While much of this isn't important, some of it is - and the trouble is that you can't tell which is which. So part of an ambassadors job is to communicate lots of tidbits which don't seem important enough for more formal communication channels.

We usually rotate the ambassadors every few months. This makes it easier for the ambassadors, who don't want to be away for too long. It also allows more people to get to know the remote team by spending time as an ambassador. In choosing ambassadors it's very important to pay attention to individual needs and preferences. Some people don't want to spend several months thousands of miles away from home, so they shouldn't be ambassadors.

We've also found that it's important for project managers to spend some time as ambassadors. Much of a project manager's job is to help resolve conflicts and flush out problems before they become serious. Experience working on both side of the telephone line is really important for them to do that effectively.

Don't Underestimate the Culture Change

One of the hardest parts of introducing agile methods into an organization is the cultural change it causes. Indeed we've found that this is the major reason why organizations have problems with adopting agile methods. Many companies operate with a command and control model which assumes that seniors make decisions and lower level people carry them out. To make agile methods work you need much more autonomy and decision making by the doers.

We find this to be a big problem in western companies, but the problem is amplified in Asia since Asian cultures reinforce deference to superiors. (A training course I saw from a major Indian contracting company defined management as "the science of control".) In this environment people are often discouraged from asking questions, talking about problems, warning about unfeasible deadlines, or proposing alternatives to perceived instructions from superiors.

The bad news for this is that getting teams to be more pro-active is an uphill battle, and one that inevitably takes a lot of time. You can never assume that problems will be raised, even when they are spotted. Getting people used to a distributed control style of management takes longer than you think.

But there is good news. Once people realize they have the freedom, and the responsibility, of making decisions - they seem to really relish it. Several of our Indian team told me how their friends in other companies cannot believe how much autonomy they are given. This autonomy is a great motivator, allowing people to be both more productive and able to grow into greater responsibility. For me one of the most interesting things we will discover is what the longer term effects are of this cultural impact, both in Asia and in the West.

Even talking about these cultural issues can cause problems. Some (western) industry analysts who saw a draft of this article said that this section was patronizing and offensive. One of our developers in Bangalore said I'm being far too mild. Another commented that it's an issue, but questioned as to whether it was worse that it is in many western companies. But there seems to be some consensus that there are cultural forces in Asia that reinforce command and control, and that this is changing.

(This is a particularly sharp issue for ThoughtWorks. We have a pronounced anti-authority attitude that's striking even in the US. We decided from the beginning that we would retain that same culture in India. I'm glad to say that we certainly seem to be succeeding.)

Use Test Scripts to Help Understand the Requirements

With greater distance, you need to put more ceremony into communicating requirements. We've been able to do that while still sticking to many of the techniques that we use in single-site development.

Increasingly I've found that more mature XP teams use acceptance tests as ways of communicating requirements. Such teams get test scripts written out before the start of an iteration to help clarify the requirements and give the development team a concrete target to aim at. One style that's worked well is for a US based customer to write a short narrative (a couple of pages) to flesh out a feature (story in XP lingo). An Indian based analyst/tester then creates test scripts for this story. This can be done either for automated or manual testing, although we very much prefer automated tests. As the scripts are developed the US and Indian analysts coordinate by email and IM as well as regular (2-3 times a week) conference calls to review the test scripts.

We've found that this has very much helped both the Indian analyst and the US customer really understand the requirements. Writing out the tests forces the Indian analyst to really understand what's needed and to ask questions of the US customer as questions turn up. The developers find it easier to ask questions of the Indian analyst rather than dig through the test scripts, so having an Indian analyst/tester is still important. Search engines are good, but humans are often easier to work with.

Use Regular Builds to Get Feedback on Functionality

When people consider the requirements gathering in agile methods, they often fail to see the importance of the feedback loop. Often the requirements process is seen as analysts providing requirements, which

developers go off and implement. At some later point the analysts check to see if the developers have implemented what they were asked for. On an agile project, the close proximity between customer and developer allows the customer to monitor progress much more frequently, which allows them to spot misunderstandings more quickly. Furthermore a partially developed system can also educate the customer, for often there's a difference between what's asked for and what's needed - and usually that's not apparent until there's some working software.

Having regular integrated builds allows a US customer to pull down last night's work and try it out. While this isn't quite as immediate as co-location, it still allows the customer to correct any misunderstandings quickly; as well as allowing them to refine their own understanding of the requirements.

To make this work, it's critical to sort out the environment issues so that you properly duplicate the environment on both sides of the ocean. There's nothing worse than onshore people pulling down a build, finding problems, and the offshore people being unable to duplicate the problem due to environment configuration issues. Make sure the environment is sorted out early, and ensure someone is around to fix any environment problems if they appear.

Use Regular Short Status Meetings

Agile methods promote regular short status meetings for the entire team (Scrums in Scrum, stand up meetings in XP). Carrying this over to remote groups is important, so they can coordinate with other teams. Time zones are often the biggest problem here, particularly between the US and India where any time is awkward for somebody. On the whole we've found that twice a week stand-ups seem to work well and provide enough coordination.

With time zone problems it's important for both sides to give and take in picking the time for calls. One of our clients would only do the calls during their business day, which forced the Indian folks to come in at very awkward hours. Pushing all the onus on one side to accommodate isn't helpful for smooth collaboration.

Use Short Iterations

In general agile methods use shorter iterations than a lot of other iterative approaches. At ThoughtWorks almost all projects use iterations of one or two weeks in length. A few of the more experienced Indian developers have worked at places which use two to three month iterations and they report that the shorter iterations are much better.

We have found that while onshore-only projects seem to be gravitating to one week iterations, the offshore projects feel that two weeks is a minimum due to the communication overhead.

Use an Iteration Planning Meeting that's Tailored for Remote Sites

On most of our projects we've found that a planning meeting at the beginning of each iteration that involves the whole team really helps to get everyone coordinated on the next chunk of work. I've noticed that most of our projects have come up with their own variation of the Iteration Planning Meeting (IPM) to suit the local circumstances. (This kind of self-adaptation is an important part of agile processes.)

A remote team adds its own set of constraints, particularly when you have awkward time zone issues. However despite the pain of awkward meeting times, we still find the IPM to be extremely useful.

Before the IPM the US customer sends narratives for each scheduled feature (story) which most projects

turn into test scripts before the IPM. During this period any questions are handled by email. Just before the IPM the development team breaks the features down into finer grained tasks. These task breakdowns are shared with the US for feedback.

All of this pre-work shortens the phone call which now concentrates on any issues that come up from the task breakdown. We find the calls usually last around a half to a couple of hours.

When Moving a Code Base, Bug Fixing Makes a Good Start

Two of our projects involved taking a large (hundreds of thousands of lines of code) code base and moving substantial development of the code base to the Bangalore lab. In both of these projects the Indian team began with a few iterations of bug fixing before they started adding new functionality.

Starting with bug fixes allowed the Indian team to become familiar with the code base, before they did substantial work on it, since bug fixes involve more code reading than changing. Although this worked well, there is some concern that more experienced people may consider it to be a stigma to be doing only bug fixes. While some people may perceive this as a problem I believe that working on bug fixes or very localized feature changes is one of the best ways to get familiar with a large new code base.

Separate teams by functionality not activity

Much of the traditional thinking on the onshore/offshore boundaries is based on the activity that people do. So analysis and design is done onshore, construction done offshore, and acceptance testing is done onshore. This obviously fits well with the waterfall model.

We've found that contrary to this, matters improve when we make the offshore team handle as many activities as possible. So we prefer to see them do as much analysis and design as possible, subject to the limitations that the requirements are coming from onshore. When we do split an effort with onshore and offshore development teams, we do this along functionality grounds rather than activities. We break the system into broad modules and let the offshore team tackle some of these modules. However unlike most groups that do this, we don't make a big effort to design and freeze the interfaces between these modules: continuous integration and weak code ownership allow the module interfaces to evolve as development goes on.

An important part of this is to grow the analyst part of the offshore team. The more someone local to the developers understands of the business, the more the development team can develop efficiently. So this means that you have to focus on growing the business knowledge of the offshore analyst. This takes time, but the local knowledge is a vital counterpart to the business knowledge onshore.

Expect to need more documents.

Agile methods downplay documentation from the observation that a large part of documentation effort is wasted. Documentation, however, becomes more important with offshore development since the face to face communication is reduced. This work is, in a way, a waste since it wouldn't be needed if the whole team was co-located. But given the offshore model, you need to do them - so they are part of the price of doing things offshore. That's a price in the time for people to write them, the added time because it's harder for people to understand many things from a document, and also a price in frustration when people are using them.

As well as documents, you also have more need for more active collaboration tools: wikis, issue tracking tools and the like. On the whole it seems that it's often better to favor tools that impose less structure, that

way the team can fit them better into how they want to work (one of the reasons that wikis can work so well.)

Whether it's documents or anything else, remember that other people's templates won't work for you, and you won't come up with the right scheme at the beginning. Make sure there's plenty of communication about the form of the documents and how well they are working. Expect to evolve the structure of documents as you learn what works best for your team.

> *There are two keys to successful documentation on agile projects. The first is finding the point of "just enough" documentation. This is difficult to determine and will vary by project. Fortunately, the iterative nature of agile development allows you to experiment until you get it right. The second key to successful agile documentation is to not get attached to it or have unrealistic hopes of keeping it updated. Documentation must be created to serve a specific purpose, and after it has served that purpose you'll all probably have more important things to do than keep updating the documents. It may seem counterintuitive, but it's often better to produce fresh documentation the next time some is clearly required. A side benefit of starting over each time you need to document part of your project is that it's great incentive to keep your documentation efficient!*
> -- [Simons]

Costs and Benefits of Offshore Development

There's still many differences of opinion, both within the general enterprise software world and within ThoughtWorks, about the costs and benefits of using offshore development. The reason why most people look to offshore is to reduce costs, noting the significantly lower rates that you find from offshore vendors. However it's foolish to look only at rates. Rates are only one component of costs, and in any case you have to look at the entire return on investment. Most people in the software industry know, or should know, that productivity differences between developers are far greater than salary differences - and even the rate differentials offered by offshore aren't necessarily greater than that. Offshore work also introduces extra costs and risks that may offset the rate differential.

The biggest consequence is the effect on communication. Offshore makes communication harder both due to the distance, which makes it difficult to meet face to face, and the timezone offset. Both of these increase the likelihood of building the wrong functionality as mis-communications occur over requirements. While techniques such as using ambassadors tries to reduce it, there's still going to be some effect. Also the distance between development and business also reduces the motivation of the development team,since they have no personal relationship to build on.

Of course a high ceremony organization that uses documents as the primary communication mechanism will not suffer as much from this. Essentially their communication has already taken all the damage from lack of direct contact, so the offshore effect is less notable. Agile methods try to restore the direct contact in order to improve communication. Our experience is that even if an agile approach suffers from the communication difficulties of offshore, it's still better than a documentation-driven approach.

Another trend may work to help with this problem. Increasingly companies are moving other business process functions offshore. If a company moves its accounting function to India, then software to support them can be built in India more easily than it could be in the west. If this kind of movement of business work offshore continues, then Indian development could become the onshore alternative.

Another benefit of offshore that's coming up is the use of 24 hour development to reduce time to market. The benefit that touted is that by putting hands on the code base at all hours of the day, functionality gets written faster. I must admit this seems a somewhat bogus argument to me, since I don't see what adding people does in India that it wouldn't do by adding them to the onshore team.

The nugget in the 24 hour development idea is that despite the tech slowdown it's still not easy to get talented developers. So often you can't get enough talented developers in the onshore location, so an offshore team is valuable for their talent rather than any lower cost.

Among all these differences my point of view is clear: I'm sitting on the fence!

The Future of Offshore and Agile

As I write this, offshore development is very fashionable, but it's still too early to really understand its true strengths and pitfalls. Certainly anyone doing because they think they'll get cost savings similar to the rate differences is seriously deluding themselves. Some people talk about all software development moving to the third world in the same way that the steel industry did, others think that after a period of fascination the offshore industry will dry up. My crystal ball just shows me what's in front of me, in a slightly distorted way.

One conclusion is clear, anyone who thinks that onshore developers will triumph because they are more skilled is very wrong. We've found that we can hire just as talented developers in India as we can in North America and Europe.

The weak spots of offshore development come from culture and distance with the business. Because agile development works best with close communication and an open culture, agilists working offshore feel the pain much more than those using plan-driven approaches. But it's still less pain than the plan-driven methods themselves!

We may never really understand the pros and cons offshore development. Software development is an activity who's output is impossible to measure. As such we'll never have hard numbers to prove one approach better than another. What we will see is growing qualitative feedback on the benefits of agility and offshore development - these qualitative assessments will determine if either, or both, will survive.

Further Reading

Matt Simons, one our project managers who has been the most involved with our Bangalore lab, has written several articles on his experiences, including Internationally Agile

Revision History

Here's a list of the major updates to this paper

- *September 2003:* First published