Abstract

Over the past few years distributed and offshore teams have been held up as the most cost effective delivery model for IT projects of all types. Yet case studies do not support these assumptions. The authors had the opportunity to compare the productivity, quality and performance of collocated agile teams with distributed agile teams and found that the collocated teams perform substantially better. In the first case a collocated team was eight times more productive, while in the second case, transferring the work to a collocated team saved the project from certain disaster. From these examples and others, there are strong indications that companies should be rediscovering collocated project teams as the old/new paradigm for delivering real value for their IT projects.

1. Introduction

There has been much industry hype about the benefits of using off-shore and distributed software development teams. Some outsourcing companies are claiming as much as 30% savings over traditional development methods. However, real metrics for quality, efficiency and productivity based on several case studies comparing collocated and distributed development teams do not support this rosy picture. This paper looks at two different cases where agile collocated teams delivered better quality work in significantly less time and with fewer resources, even more cost effectively than agile distributed teams. The cases demonstrate the difficulties of managing distributed software development teams. We found that, ultimately, excellent communication is the best predictor of success and that team collocation encourages close interactions and maximum productivity.

2. Distributed to collocated conversion

A large financial institution decided to review its assumptions about the effectiveness of the off-shore team for an upcoming major project due to previous poor outcomes for time to market and quality on a previous analogous project using off-shore resources. Since the project was to develop a high profile banking application designed to protect the company from identity theft fraud, project success was an imperative.

2.1. Original team configuration

The original estimate of work, based on similar previous projects, was for a team 20 contractors on-site and an additional 60 in Hyderabad, India. In addition, the team would use a full time project manager and part-time support from domain experts, a functional analyst and architects from the client to support requirements gathering and quality assurance.

2.2. Converting to agile

After a review of the requirements, it looked like a collocated cross functional team using Scrum/XP practices could bring significant improvements to the development efforts. The team consisted of a Scrum Master, 5 software developers, 1 functional analyst (FA), 1 Business Subject Matter Expert (SME) and 2 testers. They had support from additional business SMEs and a relational database team as needed. Note that team members were all new to each other, the technology and agile methodologies.

By applying agile methodologies and keeping the team local, the project was able to complete the work on time using only 10% (10 Full Time Equivalent (FTE) workers) of the original estimate and improve the time to market by 50%. The success of the team and the project was due to the implementation of radically different processes than anything the company had done before. The financial institution was happy with the high degree of flexibility,
improved communications, reduced hand-offs, and rapid customer feedback that resulted in a higher quality product, delivered earlier and with a smaller team.

2.3. Project barriers

To help sell the new agile model within the company, the project sponsors gave it a Pilot status with full support from upper management. This helped smooth political barriers so the team was able to rapidly commit the staff, licenses and space for a team room. The decision to exclude the Indian contractors from the pilot team caused friction with the company management, who were vested in the off-shore contractor model.

The departmentalized nature of the organization created more challenges. The customer and QA management were initially reluctant to commit full time resources to the team. The staff had little understanding of their new cross functional roles, while their managers were worried that they were losing the flexibility to task switch their resources into other work if needed.

2.4. Evangelizing agile

The following is a list of the key process and implementation improvements that allowed the 90% improvement in team efficiency. What is important to note is that the team saw itself as a cohesive whole, rather than a set of individuals who each had a specific function or role.

- Each team member was expected to be cross-functional. FA’s and SME’s found themselves testing, developers flushed out requirements and testers created customer demos for rapid project feedback.
- The core team was empowered to make all the decisions.
- The team shared work space for maximum team communications.
- All the team members had access to each other’s tools. This became especially useful as the use of Quality Center (QC) was expanded to support Test Driven Development (TDD).
- The stories (or functional requirements) were created jointly by all the members of the team, including the customer, FA, developers and testers. In fact, QC became the glue between requirements, code and test and is a key component in the implementation of effective TDD. Another big advantage of the testers being so close to the team and the process was their knowledge of each story as it changed over time, so they were better able to discover and develop negative tests.
- The stories were broken down into two week iterations by the developers themselves, who had the most intimate knowledge of how long it would take to do the work. Iterations were then planned based on the estimates.
- As the developers worked on iterations, they are able to go into QC to find the requirements (the tests) as well as any other pertinent documents. Since the customer was also on the team, any ambiguities could be discussed and resolved immediately.
- Any additional tests needed fully test the functionality would be added to QC and, in effect, update the requirements (remember tests are requirements). In this way, code, test and requirements were always 100% in sync. Since the QC document connected to QTP tests, this maintained complete requirements to test traceability.
- The stories were demonstrated to the customer giving them an opportunity to comment and make changes before too much effort had been wasted on invalid requirements. Changes were quickly reflected in modified tests updated in QC.
- Traditional unit testing was not an option, so it was decided to expand the traditional use of QTP and create more tests and increase coverage of each feature. The collaboration between the developers and testers in the team room made this possible, gaining the benefits of automated unit level regression testing using a functional testing tool against a commercial case tool.
- To keep the team on track, delivery metrics were captured during each iteration. Burn-down charts were displayed showing team iteration speed and projected next release date as well as projections for the overall work backlog.
- In effect, the phenomenal quality boost increased confidence in the product because the demos were effectively bug-free.

3. Exposing distributed team weaknesses

The second case illustrates the power of collocation to expose problems early. The project delivery was an embedded C system for a medical device application. The software needed to be extremely robust and reliable. Management was nervous because the final deadline was only 4 months away, yet no working software had been demonstrated despite much coding activity. One of use (Mark Thias) was asked to review the project status and make recommendations to insure on time delivery.

The team was composed of 50 developers distributed between two companies and three locations.
There were 13 company architects in the US and 30 company developers in India. In addition, the company hired an independent consulting company that contributed seven developers who worked at their own site.

### 3.1. Uncovering the truth

After a quick review of the previous eight months of work and meetings with the project manager and two senior architects, the real story quickly emerged. The good news was that the project had a stable set of requirements. On the other hand, it was clear that none of the work was estimated, there was no tracking of what had been completed or what was functional, or any way to determine metrics to gauge the team’s speed or productivity. The project manager claimed the project was on schedule, but since he had no working modules, there was no proof. Both architects concurred that they would have something functional in about twelve months; eight months later than management expected.

The next step was to quantify just how much work actually needed to be completed and where the productivity problems lay. Since the project had stable requirements, creating Use Cases rather than stories made the most sense as a way to break the project into manageable pieces. The work was estimated in Ideal Developer Days (IDDs) where an IDD was a perfect, uninterrupted developer day. All the developers participated in the estimation process. The initial estimate produced 2100 IDDs which was already an eye opener.

As the developers planned the first three week iteration, more reality set in. Developers were now “on the hook” to produce working modules at the end of the iteration, so they had to create a reasonable amount of work that they felt confident they could deliver within the timebox. On average, they committed to seven IDDs per developer, or 350 IDDs per iteration. So based on the 2100 IDD backlog, there was now six months of work; not great, but within striking distance of the required four month deadline.

As it turned out at the end of the first iteration, the team delivered less than 100 IDDs. Extrapolating these results would mean delivering in 21 months, completely unacceptable to the now panicked corporate management team. At this point some retrospectives started to uncover where the problems lay. The first metric that jumped out was that the 30 off-shore developers delivered less than 10% of their commitment, with five developers delivering 0 IDDs. The off-shore developers were very dependent on the on-shore team which further reduced overall team productivity. The excessive lack of productivity from the offshore team was caused by a combination of problems:

- The offshore team had limited hardware access, so they were unable to integrate and test with the same capacity as the on-shore people. Untested code was checked-in with the expectation that the on-shore team would validate and fix it. Not supplying the off-shore team with all the needed hardware and network access is unacceptable no matter what the methodology. The issue was addressed immediately and corrected.

- The off-shore team demonstrated a serious lack of programming skills. In fact, fewer than 15% of the offshore developers were competent in the programming language of choice. Despite the “on the ground” reports of poor engineering practices and lack of skilled programmers off-shore, this concern was largely ignored. It would take several more futile iterations, embarrassing customer demos and lost credibility before this issue was taken seriously.
3.2. Metrics

As shown in Figure 1: Team Velocity, the team was nowhere near the level of productivity required to meet the four month deadline. Subsequent iterations essentially demonstrated the same thing: heavy reliance on the underperforming off-shore team to deliver huge chunks of work, a subsequent failure to deliver, and a failure to respond to the reality of the situation.

Based on the data shown in Figure 1, the iteration was 30% efficient. Yet, some groups were delivering as promised as can be shown in Figure 2. The seven developer Hot Shot, Inc group represented 14% of the total team, yet was consistently assigned 18% of the total work and had a delivery rate of 78%. While the off-shore team represented 60% of the total team, was assigned 54% of the total work with a delivery rate of only 27%. The vast difference in performance efficiency was due to Hot Shot’s co-location, high skill level and comfort with agile methodologies.

Despite the clear evidence, it was very difficult for the management team to accept that their strategic directive of growing the engineering staff in India as a means of reducing cost to stay competitive was indeed flawed. Iteration after iteration, they were faced with abysmal productivity numbers yet they remained frozen with indecisiveness. In the end, work previously assigned to the off-shore team was transferred to Hot Shot, Inc. The quality of the small, collocated Hot Shot, Inc. team’s work was excellent and their productivity remained high. Contrary to the executive management’s expectations, Hot Shot’s overall cost was actually much lower compared to the off-shore team due to their vastly higher productivity and quality.

This case illustrates how agile practices challenge assumptions about the off-shore model to quickly expose a team’s weaknesses by capturing real metrics of project progress. The company management had made the assumption that their contract with the outsourcing company would ensure the Indian developers would complete the project on time and within budget. Sadly, they found to their detriment that relying on a contract for protection is unwise. Agile estimating, planning and tracking metrics were a crucial part of the project. By using agile methodologies and a seasoned collocated subgroup, the team was able to deliver the project in spec and on time, and so avert potential disaster.

4. Creating powerful teams

Clearly while the promise of off-shoring as a big cost savings bonanza is a mirage, highly productive and cost effective software development is not only possible, but achievable, by using agile techniques and investing in collocated development teams. To achieve the highest level of productivity and communications, creating collocated teams is the preferred option. Some of the factors that really make a collocated team so productive are:

- Customer/developer partnership
- Cooperation through collocation
- Short development iterations
- A focus on quality
- Eliminating waste through learning

4.1. Customer/developer teams
In a distributed team model, often the development team is not only not the business owner, but not even part of the same company. As a result, the developers have no sense of ownership of the product nor do they share the same fundamental goals. The sponsoring company is interested in a quality product that meets their requirements, while the consulting company is interested in delivering the product as cheaply as possible with little or no regard for quality or future maintainability.

One of the biggest advantages to the agile approach is that the customer works closely with the teams – in fact becomes a full functioning member of the team. Each development iteration gives the customer (whether internal or external) the opportunity to verify that the requirements are met as the project develops. The customer also has the opportunity to evaluate whether they have provided the right requirements, because they can quickly see the results of their input to the product. In this way, defects in requirements are exposed and corrected early.

4.2. Cooperation through collocation

The most difficult hurdle to get over when using a distributed development team is the problems of distance and other related risk factors. With modern communications and the ubiquitous Internet, it might seem that the need for face to face communications can be eliminated. However, many studies have shown the effect of regular personal communications on productivity. In contrast, a high functioning collocated team will be in continuous communications. Not only are there daily check-in meetings to catch any issues as they arise, but the team is often working in a shared environment. It is much easier to consult with a co-worker on a question about some piece of code when that person is sitting in the next cubicle. The value of close proximity for improved productivity cannot be over stressed.

4.3. Short iterations

A typical distributed offshore team will generally deliver software in six month increments. Since there is little or no insight into the inner workings of the team during the long development cycle, it is obvious how defects and waste can easily become hidden within the process, resulting in low quality software deliverables.

In contrast, collocated teams are grounded in practices that consume and implement improvement ideas at every turn. Software is broken down and delivered in 2-4 week iterations with a retrospective or kaizen session being held at the completion of each iteration module. Progress is measured in terms of tested, working software, not time or code completed milestones. During each iteration defects and waste are exposed which leads to the highest quality and lowest cost. Retrospectives generate ideas for improvements which can be implemented right away in the next iteration. To keep the project on track, a light set of metrics is created and applied to determine the benefit of each new idea as it is incorporated into the product. Over time, the team is more productive while the software quality improves.

4.4. A Focus on quality

The unit testing practices of typical distributed teams are often not as rigorous as collocated teams because they are not part of the core development process. Most consider unit testing to be a delay rather than the quality improvement tool it is. Since most distributed teams have split the QA function to another group, testing is not a high priority, so all the actual testing is left to the functional testing team, which is not focusing on the code itself. This can reduce code review coverage, sometimes to as little as 10%-15%.

Collocated software teams focus on quality by developing automated unit tests for each and every function in the software. This has the side effect of reducing wasted effort by automating the build and integration process and ensuring a bug free demonstrable system on a daily basis. Since the product remains functional throughout the development cycle, the team has greater flexibility to respond to changes in requirements and functions that come from the business owners and the team owns an understanding of the inner workings of the software, a key component to future continuous learning and improvement ideas.

4.5. Eliminating waste through learning

Distributed teams gain no advantage in sharing ideas realized during the development process. Offshore teams are accustomed to being handed completed requirements, a defined architecture and are expected to do exactly what they are contracted to do. In fact, the climate and culture of distributed teams are to get the work done as fast and as cheaply as possible generally by throwing resources and time at every problem. Due to these factors, their participation in improvement ideas is often fruitless or even irrelevant since they have little insight into the business requirements of the deliverables. To them, the long-term quality and maintainability of the software is not a
priority, since they will be long gone before any problems will crop up.

Collocated organizations, on the other hand, are always looking for ways to continuously eliminate waste and constantly implement improvement ideas. The best people to identify waste and generate ideas for continuous improvement are the people doing the work, of course. Stakeholders can easily go to the software developers, the architects, the testers and ask for new innovative features or new technologies that can help gain a competitive advantage. While programmers are engaged in their work on one project, lots of creative ideas are realized that can be used immediately or for other projects. In this way, a company can maintain an innovative backlog of product ideas.

5. Conclusion

The underlying assumptions about distributed teams’ relative labor costs, skill sets and productivity are now shown to be incorrect and increasingly less justifiable. To combat these trends and reverse the rapid decline in productivity and quality, it is time to take back the software development process by applying development methodologies that allow better team communications, faster delivery times and higher quality products by using agile practices. By applying these principles along with the creation of continuous improvement and continuous learning environments, it is possible to create small, powerful, adaptive, sustainable software teams that far out perform any off-shore or distributed development team.