Communications of the Association for Information Systems

Volume 23 Article 34

12-2008

Lessons Learned from Distributed Agile Software Projects: A Case-Based Analysis

Indranil Bose
The University of Hong Kong, bose@business.hku.hk

Communications of the Association for Information Systems

CAIS

Lessons Learned from Distributed Agile Software Projects: A Case-Based Analysis

Indranil Bose

School of Business

The University of Hong Kong

Pokfulam Road, Hong Kong

bose@business.hku.hk

Abstract:

Agile software development in a distributed setting is challenging. The teams involved in the process face difficulties in communication, personnel selection, work culture, and knowledge management. The shortcomings associated with working in different time zones and the inabilities to develop trusting relationships between developers are well known. Companies often take recourse to agile software development methods in a distributed environment in search of reduced cost, higher efficiency, increased flexibility, and good customization. However, it is not clear whether agile methods can be successfully followed and their benefits realized in a distributed setting. This paper revisits and synthesizes the lessons learnt from twelve case studies detailing successful implementation of distributed agile software projects. The cases are analyzed from the perspective of the agile manifesto to determine how closely they follow its values and principles and to what extent they realize the benefits of the agile methodology. The cases lead to the discovery of disparate and innovative solutions adopted by different companies for overcoming the challenges of distributed agile software development. Some solutions are commonplace and others are unique and their combination in the context of the challenges is enlightening. The list of solutions can suitably guide companies that plan to adopt the agile methodology in distributed software development environments in future.

Keywords: agile methods, case analysis, case studies, distributed development software development

Volume 23, Article 34, pp. 619-632, December 2008

I. INTRODUCTION

Distributed software development has been around for some decades. Many large organizations have taken advantage of it for projects that involve cooperation and collaboration between teams located at different locations. In distributed software development the bulk of the work is often offshored to developing countries with a small team of consultants working onshore with the clients. Other than low cost, the main drivers for distributed software development are flexibility and increased productivity. The lure of distributed software development arose from the availability of talented knowledge workers in multiple locations around the globe, who could develop software of high quality at lower cost. Distributing the work also meant that developmental work could be done around the clock taking advantage of the difference in the time zones. But managing a distributed software development team is a great challenge. This is due to complexities resulting from asynchrony of communication, differences in work culture, and differences in organizational practices. Companies that look for economic methods for developing quality software sometimes combine distributed development with agile methods for software development. The hope is that such a combination will allow companies to leverage the advantages of agile methods and increase the gains achieved in distributed software development. However, incorporating agile methodologies in a distributed setting makes the software development process even more challenging. In this paper, we analyze and synthesize twelve case studies on successful implementation of distributed agile software projects that address the key challenges encountered in such projects. The solution strategies proposed are listed in the context of six commonly occurring problems in such projects and studied from the perspective of the values and principles of the agile manifesto.

II. AGILE METHODS OF SOFTWARE DEVELOPMENT

The agile method of software development aims to develop software quickly, economically, and efficiently. From the late '90s, the agile methods have become popular because they can take care of volatile customer requirements, establish close interaction between customers and developers, and deliver software within shorter time periods and stringent budget constraints. Various agile software development methods such as extreme programming (XP), feature-driven development, crystal clear method, and SCRUM, etc., have become popular in recent times [Abrahamsson et al. 2003]. In a traditional plan driven model software is developed in a sequential manner—requirements are gathered from clients, different roles are assigned to developers for coding and control, and finally software is tested and integrated with existing systems [Huo et al. 2004]. In doing so, a large amount of documentation is produced that contains knowledge about developmental processes. Interactions between clients and members of the development team are very little, and the focus is on achieving efficiency by improving the repeatable processes.

Agile methods aim to overcome the limitations of the plan driven approaches by considering that requirements for building software are not static but dynamic. Hence, instead of development taking place in the form of highly defined processes, it takes the form of "minimally defined and adaptive" processes in agile methods [Bowen and Maurer 2002]. They also aim to involve customers in the software design from the very beginning for providing feedback about the product and the process. Since the customers know how the software is shaping up from the beginning, they have the opportunity to delay decisions on certain items till they become absolutely necessary. This helps reduce costly overruns at the end and helps the code remain up to date with the requirements. Also, as the requirements keep changing, there is less importance attached to keeping proper and lengthy documentation. Agile methods use the repetitive process of short iterations consisting of build, then consult with customers on site, and then build some more to avoid mismatch between the deliverable and the customers' expectations. Some experts have compared this style of software development to "artful making" because of the lack of structure in the entire process [Austin and Lee 2003].

Agile methods also tend to build software with teams of smaller size, and there is frequent face-to-face interaction between team members. A commonly used term in agile development is *pair programming*, where two expert programmers work on the same piece of code to improve its quality. While the benefit of pair programming is constant inspection of code quality, it can only succeed when there is understanding between programmers in terms of accepting criticisms, suggestions, and ability to follow each other's instructions. That is why some authors have called the agile methods more people driven and less process driven [Nerur et al. 2005]. Another important feature of the agile methods is refactoring where the programmers improve the internal structure of the code without changing its outward appearance. The impact of these changes is usually small but they go a long way in improving software quality. Figure 1 shows the different steps followed in the agile method of software development.

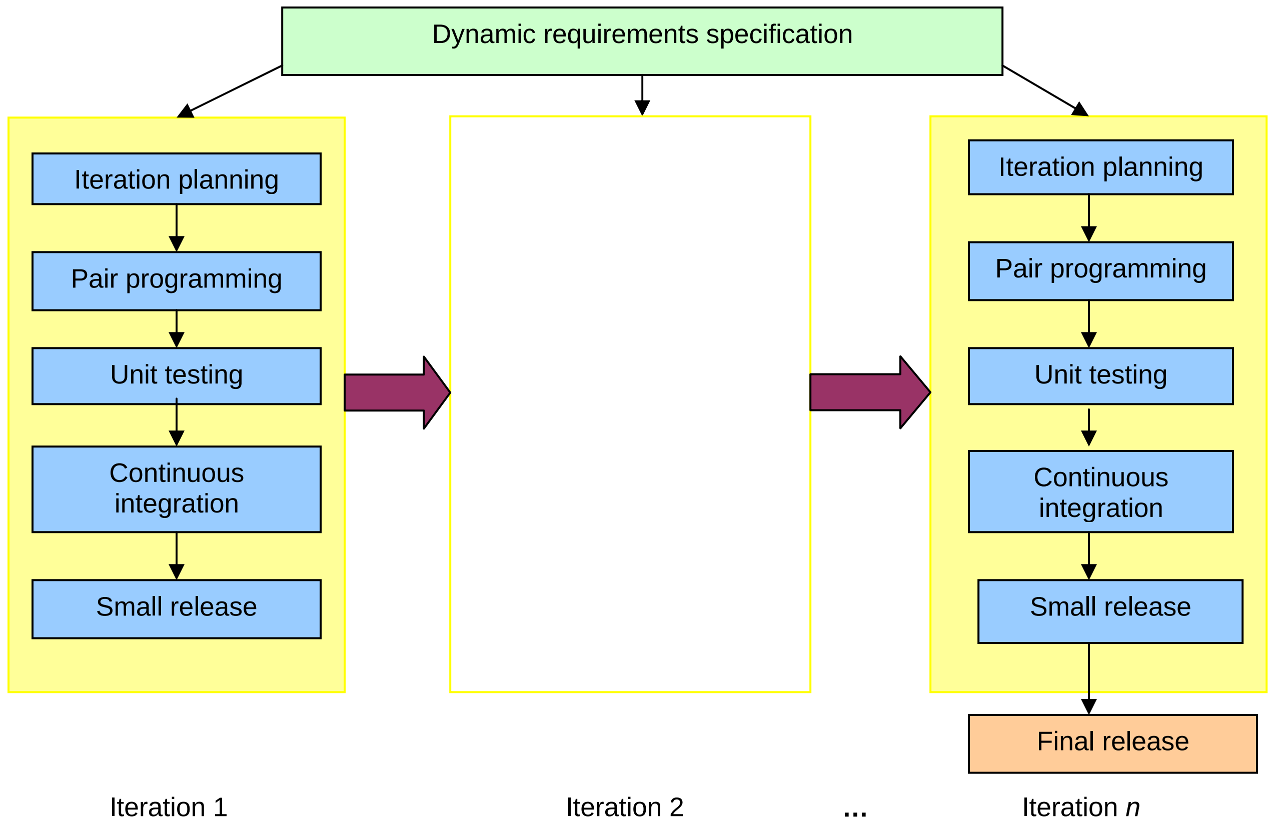

Figure 1. Different Steps in the Agile Method of Software Development

In terms of testing and integration, the agile approach is very different from the plan driven approach [Talby et al. 2006]. The agile method often follows a test-driven approach. Once a portion of the code is written, it is released as an early version to the customer, integrated with the customer's existing system, and tested to identify the avenues for improvement. This is known as *continuous integration,* and it makes the process of searching for bugs cheap and efficient. Testing in agile methods usually occurs much earlier in the developmental life cycle. Early customer feedback is a key feature that enables modification of requirements from time to time and provides a form of acceptance testing. The key features of the agile methods of software development and the benefits of these features are listed in Table 1.

Table 1. Key Features of Agile Methods and Their Benefits

Features	Benefits
Continuous requirements gathering	Customers delay decisions about crucial items; software remains flexible
Frequent face-to-face interaction	Overcomes misunderstandings; builds trust among team members
Pair programming	Easier teamwork; better ownership of code
Refactoring	Gradual improvement of code without creating a shock wave
Continuous release and integration	Detection and fixing of bugs earlier on in the project; higher software quality
Early expert customer feedback	Avoidance of costly overhauls in the end; lower cost of development
Minimum documentation	Smaller development time; lower cost of documentation

III. AGILE METHODS OF DISTRIBUTED SOFTWARE DEVELOPMENT

Agile methods are well suited when customers and developers are collocated and there is frequent interaction between the two groups [Boehm and Turner 2005]. However, there is increasing evidence from practice that indicates that agile methods are being adopted by small as well as large organizations. For example, organizations like ABB, Daimler Chrysler, Motorola, and Nokia have all adopted and recorded their positive experiences in using agile methods [Lindvall et al. 2004]. Among offshore organizations embracing the agile methods Cognizant Technology Solutions and ITC Infotech India are the most prominent. Vendors like ThoughtWorks and Valtech have even paved the way for successful distributed agile development [Moore and Barnett 2004]. However, there is no denying that incorporating agile methods in distributed software development requires considerable effort. This section describes the various challenges involved in this adoption and some generic considerations and possible strategies for addressing these challenges. The solution strategies suggested are not exhaustive by any means. In the following section, 12 case studies are explored in more details to find out what specific solution strategies are actually adopted by those companies to overcome the challenges.

Communication

Developers need to communicate frequently and teams located across time zones make this difficult. There is also the need to have effective communication with active customers to get their feedback about the quality of the product in progress and the changing requirements of the project. Onshore and offshore teams often use unstructured wikis to communicate asynchronously about the status of the project [Fowler 2004; Sepulveda 2003].

Personnel Selection

Programmers who are used to the planned approach and adept at following orders of the project managers are not a good choice for the agile team. Since some agile methods prefer pair programming it is essential to find coding partners who have similar mindset. Also, as requirements keep changing in an agile project it becomes necessary to have a cross-functional team consisting of analysts, developers, testers, and project managers. The members also need to be able to communicate with customers and understand their needs.

Work Culture

Offshore teams often use plan driven development. It takes time and patience to change this culture. There is a need to educate developers in offshore teams about agile methods through cross-team workshops and informal meetings [Fowler 2004; Sepulveda 2003]. Also, in order to inculcate independent thinking and action, the traditional incentive and rewarding schemes need to be modified. Onshore team members must be aware of prevalent work practices and customs of offshore locations (e.g., absences during religious festivals) [Nisar and Hameed 2004].

Different Time Zones

Agile methods emphasize frequent face-to-face meetings that are often arranged at a short notice. This is impossible for distributed teams as the hours of work are different. To overcome this it is important to instill a good build discipline among developers so that the finished build of the software at the end of the day can be downloaded and tested by the onshore team and even by participating customers [Fowler 2004; Nisar and Hameed 2004]. When meetings are arranged it is imperative that both teams are allowed to set their preferred meeting times.

Trust

Trust is extremely difficult to achieve in a distributed setting [Sepulveda 2003]. Frequent meetings have to be substituted by seeding (when initial requirements are conveyed) and maintenance visits (status checks) by members of the onshore team and even by customers, if possible. These visits must be short and informal so that they are not looked upon as policing visits. Also, project managers need to act more like facilitators than commanders-in-chief to encourage independent thinking among offshore team members.

Knowledge Management

Agile methods' emphasis on "just enough" documentation may not work for distributed environments where developer turnover is high [Bowen and Maurer 2002]. If tacit knowledge of programmers is not documented, it won't be reused. Wikis, issue tracking tools, and remote screen capture methods may be used for documentation and documentation templates must be reviewed from time to time to make sure they are working appropriately. Code repositories are also an essential part of the knowledge management effort [Rees 2004].

IV. CASE ANALYSIS

Due to challenges facing the deployment of distributed agile software development, it is not surprising that only few case studies exist that discuss the experiences faced by the onshore and the offshore teams. In order to learn from the experience of others, a search was conducted for case studies that followed the definition proposed by Yin: "A case study is an empirical inquiry that investigates a contemporary phenomenon within its real-life context" [Yin 2002]. This resulted in 12 case studies that claimed to be successful in agile software development in a distributed setting. These cases are published as either journal or conference papers. They study the challenges faced in implementation of software projects in the context of distributed agile development. Other case studies documented in this area are not included because they either lack details or do not dwell on innovative solution strategies. Table 2 provides a list of the software companies that undertook distributed agile software projects with offshore teams located in other countries. It is seen that among the 12 cases, 10 onshore companies were located in the U.S. and seven offshore companies were located in India. A variety of software projects became candidates for distributed agile development and no patterns were seen. XP and SCRUM were the agile methodology of choice for most cases.

Table 2: List of Examples of Successful Distributed Agile Projects

No.	Onshore Location	Offshore Location	Client Location	Software Project	Team Size	Agile Method	Reference
1	Extol International, USA	Elegance Technologies, India	UCCnet, USA	Domain-specific front-end for Extol Business Integrator	2-3 onshore and 5-10 offshore	SCRUM and FDD	[Kussmaul et al. 2004]
2	Iona Technologies, USA	Iona Technologies, Dublin, Ireland	Not specified	Application Server Platform and Web Services Integration Platform	130 engineers	XP	[Poole 2004]
3	Valtech, USA	Valtech, India	Not specified	Not specified	Not specified	SCRUM and XP	[Danait 2005]
4	Telco, USA	India	USA	Unspecified pilot projects	16	Not specified	[Balasubramaniam et al. 2006]
5	Manco, USA	India	USA	Extend functionality of complex supply chain system	14	Not specified	[Balasubramaniam et al. 2006]
6	Consult, USA	India	USA	CRM system	15	SCRUM like process	[Balasubramaniam et al. 2006]
7	Aginity LLC, USA	Ukraine	Not specified	Business intelligence application	3 groups	Not specified	[Armour 2007]
8	BNP Paribas, France	India	Not specified	Security services	100 with 50 onshore and 50 offshore members	Not specified	[Massol 2004]
9	Finnish company, Finland	Not specified	Several customers	Not specified	Teams of 6 developers each	SCRUM	[Paasivaara and Lassenius 2006]
10	WDSGlobal, UK	USA and Singapore	Not specified	Refinement of mobile configuration platform	Not specified	XP	[Yap 2005]
11	CEInformant, USA	India	USA	J2EE compliant solution software for insurance business	2 offshore teams of 6-8 members each	SCRUM	[Computer Enterprises 2005]
12	SirsiDynix, USA	Starsoft, Ukraine	Not specified	Integrated library system	36 onshore and 20 offshore members	SCRUM	[Sutherland et al. 2007]

The cases were analyzed to discover what steps were taken by the respective companies to overcome the six challenges detailed in the earlier section. Although most cases were sufficiently detailed, they did not provide information about innovative solutions for overcoming each of the challenges. Table 3 lists the various solutions that served these companies well and contributed to the success of the projects.

Table 3: Solutions Adopted by Companies for Facing Challenges in Distributed Agile Development

No.	Company	Communication	Personnel Selection	Work Culture	Different Time Zones	Trust	Knowledge Management
1	Extol International	Status e-mail to mailing list Synchronous (Web conferencing) and asynchronous	2-3 onshore staff and 5-10 offshore staff One consultant spent several days onsite	Avoid direct questions that are looked upon as challenge to authority Tolerant to	15-30 minute meetings via IM Web conferencing and phone meeting times	Exchange of staff members between two locations Onshore team more involved in design and	CVS repository to store all requirements, designs, source code, and related

		(email) communication		offshore team accustomed to the SW-CMM model	rotated for convenience Round the clock cycles by sending requirements to offshore team at end of day	offshore team in implementation Frequent delivery of software	documents Small number of documents
2	IONA Technologies	Unstructured Wiki pages used to present distributed story board Twice weekly meetings replaced daily standups Conference call during product delivery	Disparate group of engineers located across the globe	Rotate one or two developers across sites Attention paid to differences in perception of authority and body language	Daily conference calls to identify who needs to pair with whom Automated nightly build, integration, and testing	Access to common build environment Rotation of engineers Dissenting members allowed to compete or asked to leave or compromise sought	Common source base and multi-site control system used
3	Valtech India	Twikis used IM used between developers in overlapping time zones	New team member introduced by Web conference	Cross-functional kickoff meetings Flat organizational structure Role swapping allowed Coffee and ice-cream socials to celebrate milestones	Web conferencing frequently used Web kickoff meetings recorded and played back Pair programming conducted through Web conferencing Programmers did not leave office till last code checked did not break build	Pair programming used for initiation of new members with experienced members acting as mentors Chocolates distributed when someone broke build Cross-location visits implemented	Web conferencing used for virtual white boarding Burn-down charts used Bugzilla bug repository used Twiki used as a central repository
4	Telco	Informal communication took place through formal channels Short morning meetings held for status tracking, ideas and comments generation SMS and online chats used for communication Videoconferencing used by senior managers every week for handling critical issues	Project leads played a major role in co-ordination A cohesive team was built using members who had prior experience of working with each other	No particular steps taken	Meetings took place at early mornings for onshore team and late evenings for offshore team	Customer delegates spent significant amount of time with developers Senior members of management visited developer sites to evaluate progress of project	Short use cases and user stories took the place of detailed use cases A database tracked project status, notified issues, and handled priorities
5	Manco	Project lead responsible for facilitating communication and reducing miscommunication Videoconferencing used by senior managers every	Project leads played a major role in co-ordination	No particular steps taken	Meetings took place early mornings for onshore team and late evenings for offshore team	Frequent and long visits to customer sites by one or more developers Senior members of management visited developer sites to evaluate progress of project	Short use cases and user stories took the place of detailed use cases A database tracked project status, notified

		week for handling critical issues					issues, and handled priorities
6	Consult	Project leads on call via Blackberry	Project leads played a major role in co-ordination A high performance team was built using members who had prior experience of working with each other	No particular steps taken	Meetings took place early mornings for onshore team and late evenings for offshore team	A group of analysts and developers always present at the customer site Senior managers visited developers at the beginning of the project to finalize terms of contract and set up ground rules to be followed	No particular steps taken
7	Aginity LLC	IM frequently used Open source groupware used to manage development space	Choice of offshore location restricted to a country with similar culture Chosen team members are creative and intelligent	No particular steps taken	Builds, tests, and test results made visible due to lack of social communication between teams	Presence of a friendly contact who acted as an agent between the offshore and onshore team Offshore team given plenty of freedom to develop solution Iterative development using common development language and common vocabulary	Use of compelling user stories rather than dry specifications of requirements Project Web pages created to give an idea of project progress and status A variety of tools used for task and time tracking, issues control, and code management
8	BNP Paribas	Weekly technical and management conference calls with all teams Establish communication channel at all levels E-mails, chats, and Wikis frequently used Phone calls and videoconferencing moderately used	Mostly senior members made up the offshore team Project leads played a major role in business conception and detailed design	Good mediators helped in solving language and cultural problems	A lot of visits organized in both directions	Same roles used at both locations – managers, project leads, lead developers, developers Dedicated offshore support persons in each team No micro manage-ment of teams Sharing of activities like business conception, detailed design, and testing	Sharing of use cases through a repository Use of a shared quality dashboard showing results of tests Knowledge sharing (both software and culture related) at all levels
9	Finnish company	Electronic communication using mailing lists, Wiki, IRC, IM, Skype, teleconferencing. Sprint demos through videoconferencing and desktop sharing	System architect played the role of proxy customer to offshore team since the actual customer was not available	No particular steps taken	Communication is made using tools so that they remain visible	Members of offshore team invited to work at onshore location Frequent communication between teams	Project documents stored in Wikis
10	WDSGlobal	Video conferencing used during daily handover and during customer requirements gathering	An XP coach hired at each location to train people on XP and object oriented programming	Cultural differences created misunderstanding s and solved using round-the-world program	Daily handovers took place between teams when status updates as well as lessons learnt	Office space reconfigured and made cube-less in order to put pairing stations in place Offshore workers	A new source control system put into place A monthly product backlog called

		Coaches communicated with each other after daily handovers, once per week via teleconferencing and also visited each other for two weeks every quarter	with Java and work closely with business managers A rotating guru from the onshore team helped in setting up the infrastructure and initial training	where members visited each location for 4-6 weeks Some controlled flexibility to adapt processes according to local culture	were shared Configuration changes of the shared control system synchronized automatically between locations	spent several weeks with onshore team in a programming boot camp Frequent pairing used for solving complex design problems in a collaborative manner If fixes could not be completed on time work passed to the next region to "put out fires together"	Company Program Plan created for better coordination between teams and prioritization of projects
11	CEInformant	Daily focused SCRUM meeting lasting for at most 30 minutes Weekly status meetings to understand risks faced by offshore teams Wrap-up meetings for demonstration of sprints	People with a diversity of skills in various activities included in the project Product manager responsible for keeping track of and updating project backlog Team leads responsible for keeping track of sprint backlog	No particular steps taken	No particular steps taken	Offshore team had full authority to do things that were necessary for the success of the project Onshore team trained offshore team in the use of various tools and in orientation of newly recruited developers	Offshore and onshore teams used the same StarTeam repository for issue tracking Errors in coding recorded using Bugzilla Requirements recorded in terms of use cases, user interface prototypes, and standards documents Discussion threads documented
12	SirsiDynix	Daily SCRUM meeting by teleconference preceded by email communication Daily standup meeting at offshore locations prior to SCRUM meeting	Teams built according to functional areas of systems Product owner responsible for co-ordination of requirements and team management	No particular steps taken	Daily SCRUM meetings held at 7:45 a.m. for onshore team and 17:45 p.m. for offshore team for mutual convenience	No particular steps taken	Global build repository used Issues tracking and project management software used to generate real-time view of project including bugs report

V. AGILE MANIFESTO AND THE CASE STUDIES

One of the popular benchmarks for judgment of agile projects is the agile manifesto which aims to "uncover better ways of developing software by doing it and helping others do it" [Fowler and Highsmith, 2001]. This manifesto was put forward by a team of software development experts and laid down the values of the agile methodologies and the principles to be followed to attain those values. It was important to analyze the twelve agile software development cases in the light of the agile manifesto since this is a well accepted benchmark to judge agile projects. The following paragraphs describe how the twelve case studies either conformed or deviated from the values and principles of the agile manifesto.

The agile manifesto consists of four values that appear in the form of two part comparative phrases outlining in the first segment what is emphasized in an agile development project and in the second segment what is important but is of lesser priority. The values and principles of the agile manifesto are listed in Table 4 together with a check mark (✓) to indicate which of the case studies actually showed an evidence of the value or a principle. An x indicated that the case study did not support the value or the principle. It is easy to imagine that since the case studies are not detailed enough, some of them did not provide enough or no information about whether the values or principles were maintained or not. This is indicated by a -- in the corresponding column. All case studies emphasized that the projects revolved around people rather than processes. The team selection was done carefully and leaders were chosen who could coordinate the development activity. Various modes of electronic communication were adopted for frequent interaction and exchange of ideas. In the same fashion, the focus on working software also could be seen in most case studies. The agile manifesto prefers working software over detailed documentation but does not negate documentation completely. In the case of Extol International, the developers stated the importance of documentation and remarked that “key documents help to bootstrap the project by establishing a common framework for stakeholders” [Kussmaul et al. 2004]. Four case studies emphasized the importance of customer collaboration. A particularly interesting case in this regard is that of the Finnish company that was developing the software for several customers and used the system architect from one of its customers to act as a proxy customer who could talk about likely technical and business issues. The fourth value of the agile manifesto had the least support from the case studies. Most case studies did not discuss in details whether the software development process involved responding to changes requested by the customer even at the last moment. However, the case studies on Extol, Manco, and Consult clearly stated that they deviated from the agile practice of not following a plan in the early stages of the project and used a few iterations in the project to confirm the key requirements for the project as well as an architecture for the project. In fact, it was stated that “instead of strictly following the agile development practices as commonly defined, the companies continuously tweak them to fit the evolving needs of their projects” [Balasubramaniam et al. 2006].

Table 4: Adherence to the Values and Principles of the Agile Manifesto

	Case studies											
Characteristics promoted by the agile manifesto	1	2	3	4	5	6	7	8	9	10	11	12
Values												
Individuals and interactions over processes and tools	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Working software over comprehensive documentation	✓	✓	✓	✓	✓	--	--	✓	✓	✓	✓	✓
Customer collaboration over contract negotiation	--	--	--	✓	✓	✓	--	--	✓	--	--	--
Responding to change over following a plan	x	--	--	--	x	x	--	--	--	--	✓	--
Principles												
Early and continuous delivery of valuable software	✓	✓	✓	x	x	x	--	✓	--	--	✓	--
Welcome changing requirements even late in development	--	--	--	--	--	--	--	--	--	--	--	--
Deliver working software frequently	✓	✓	✓	--	--	--	✓	✓	✓	✓	✓	✓
Business people and developers work together throughout project	✓	--	--	✓	✓	✓	--	--	✓	--	--	--
Build projects around motivated individuals and support and trust them	--	--	--	✓	✓	✓	--	✓	--	--	--	--
Face-to-face conversation within the development team	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Working software is the primary measure of progress	✓	✓	✓	✓	✓	✓	✓	--	✓	✓	--	✓
Promote sustainable development to maintain constant pace indefinitely	--	✓	✓	--	--	--	--	--	--	✓	✓	✓
Continuous attention to technical excellence and good design	--	--	✓	✓	✓	--	--	✓	--	✓	--	--
Simplicity is essential	--	--	--	--	--	--	--	--	--	--	✓	--
Self-organizing teams	--	--	✓	--	--	--	✓	--	--	✓	✓	--
Team regularly adjusts behavior to enhance effectiveness	--	--	--	--	--	--	--	--	--	✓	✓	--

The 12 principles laid down in the agile manifesto were observed to some extent in the case studies. All companies enabled multiple forms of electronic communication between the offshore and onshore teams to simulate face-to-face conversations. Although these were face-to-interface conversations using various electronic technologies but they were found to be quite effective and were used on an as needed basis. Many case studies emphasized availability of working software that was usually put in a shared repository. This working software also served as a measure of progress made in the project. However, not all companies that valued working software went on to deliver such software to their customers from time to time. Only five companies mentioned doing this. Another common characteristic seen in five case studies was the strong emphasis on achieving good software quality with bug fixing taking place continuously and serious efforts being spent on not breaking the builds. A constant and sustained pace for development with pre-specified times for deliveries of builds were seen in five case studies as well. The collaboration between business people and developers took place in five case studies and only four case studies identified key business people who played a crucial role in coordinating and directing the project. One of the important principles of the agile manifesto is the availability of carefully selected teams that were mostly trusted to follow their own procedures to develop the software in a way that they felt was most suitable. Existence of such self-organizing teams was reported in only five case studies. Only two case studies indicated that efforts were spent on regularly adjusting the working mode according to need and one case study discussed the importance of simplicity of design. Finally, there were no case studies that provided details on the responsiveness of the development team to the changing requirements of the customers. Although it is likely that given the small scope of description of the case studies, they did not delve into the details of the values and principles of the agile manifesto, but still it can be said that not all values and principles listed in the agile manifesto were considered to be important in the case studies under consideration.

VI. LESSONS LEARNED

Some interesting lessons can be learnt from the commonality analysis of the 12 case studies. As suggested by the literature, a variety of means of communication was used by the teams ranging from e-mail to chat to teleconferencing. Videoconferencing seemed to be the most commonly used means of electronic communication with five cases reporting the use of it. Since video conferencing is expensive it was mostly used for the most critical parts of the project. An example is the use of videoconferencing for sprint demonstrations by the Finnish company. In addition to videoconferencing, Wikis and IM were used in four cases. Another interesting feature was the prevalence of daily focused Scrum meetings for many companies. All case studies emphasized frequent communication between the developers using a variety of technologies. However, the lack of co-location often made this communication face-to-interface rather than face-to-face. Also, the communication was not always continual in order to save costs and was done on an as needed basis.

In terms of personnel selection for the project, it is observed that different companies adopted different policies with success. Many cases acknowledged, either directly or indirectly, the importance of the presence of a strong project lead who could coordinate the various activities of the project. When selecting team members, the stress was on selecting programmers that were senior, creative, and had prior experience of working with each other. SirsiDynix was quite unique because it formed teams according to the functionalities of the system that was being built. No other company seemed to follow this practice. Also, it was observed that five companies chose to have self-organizing teams that worked well with little supervision and enjoyed responsibilities of managing their own business.

Although cultural issues are considered to be quite important a challenge, seven out of the 12 companies chose not to address this challenge. For those cases where cultural differences turned out to be problematic, the presence of a mediator was sought and more flexibility in the development was allowed. At least in two cases developers were rotated across sites so that they could learn the social and work culture of all sites involved in the software development.

Working across different time zones was acknowledged to be a major challenge by most companies. In order to take advantage of the “follow-the-sun” philosophy, certain compromise had to be sought. In five cases, the meeting time between the onshore and offshore teams was selected according to mutual convenience. The meetings were also conducted using Web conferencing so that a record of the discussion could be kept and referred to by developers who could not attend the meeting on time. The offshore team was also given flexibility in adopting the processes and tools that they felt comfortable with. However, some control was still exercised by forcing them to abide by the global standards.

On-site development ensures plenty of social interactions that ultimately lead to building of trust among developers. Special efforts towards developing trust in offshore agile development included rotation of staff members between onshore and offshore locations (used in five cases) and abundant flexibility allowed for the offshore development team (used in three cases). Valtech India was unique in building trust by allowing new members to learn about the project through pair programming with more experienced members in other locations.

There were several companies that used a global build repository that could be shared between locations. This eased the development task to a great extent. At the same time project use cases were shared between developers in some case studies and error tracking tools like Bugzilla was used in two cases. These leads us to believe that although the emphasis was on minimum documentation for agile software development the teams resorted to sharing the knowledge by electronic means as and when there was a necessity for doing so.

The case analysis identified several common characteristics in distributed agile projects that would have been seen in any distributed plan driven projects as well. For example, the need for communication via a variety of electronic means is essential for all distributed projects. The same also holds for adjustment of meeting times according to the mutual convenience of the development teams. Both these actions are essential for success of any distributed software development project and it was not surprising that they made their appearance for distributed agile case studies as well. So the agile case studies did not show any unique characteristics in terms of “communications” and “different time zones” factors.

However, some unique characteristics were seen from these case studies in terms of “personnel selection,” “trust,” and “knowledge management” that could be attributed to the agile nature of these projects. The strong focus of agile projects on individuals and interactions, as discussed in the agile manifesto, could be evidenced in these case studies as well. The offshore team was carefully chosen and preferences were given to the selection of developers who had worked with each other on prior projects. The implicit understanding was that if they had worked successfully on a prior project they understood each other’s working styles, trusted each other, and could again work together to make the project successful. In some sense, it was assumed that prior co-workers will have good interactions with each other. Also, some case studies also emphasized the need for a project leader who acted as a pivot and coordinator for the project. (S)he traveled frequently between offshore and onshore locations and often spent a significant amount of time at the customer premise. This was necessary to maintain the interaction between the various teams and also to convey on a firsthand basis the clients’ changing needs to the development teams. The project rotated around such a senior team lead that was also responsible for ensuring that there was no project backlog. Another characteristic that was seen in five cases was the flexibility that was offered to the offshore teams. This was probably done to make the teams self-organizing. There was little micro-management and teams were trusted and allowed to work on their own. They were not told what processes were to be followed or what particular tools were to be used. Also, the emphasis on working software was maintained in several cases. Usually global build repositories were constructed where the latest form of the working software was always available to the development teams. It should also be mentioned that some companies did not focus on early delivery of software but delivered software only after a few rounds of development when they felt comfortable with the current version. However, contrary to the popular belief that agile projects did not need any detailed documentation, most of these case studies showed that developers opted for moderate documentation that kept track of progress status and priorities for the projects. The focus was on working software for sure but documentation was not neglected completely.

Some important lessons can be learned from the analysis of the case studies from the perspective of the values and principles of the agile manifesto. It is observed that some values such as the importance of people and their interactions and the availability of working software were clearly followed in most case studies. Agile software development thrives on formal and informal interactions between people. This is necessary for building trust and for understanding developers’ coding habits. Multiple forms of interactions through IMs to Wikis to videoconferencing were used in these case studies. There were two components to these interactions—some were preplanned as seen in the focused SCRUM meetings organized by CEInformant or SirsiDynix through teleconferencing and some were instantaneous on an as needed basis as seen in the use of IM and chat by Aginity LLC and BNP Paribas. Some companies like Valtech encouraged onshore and offshore developers when they came together to take part in informal social gatherings like coffee and ice cream socials to build trust and friendship, while celebrating completion of milestones. This practice was unique but in conformance with literature that stressed the importance of “renewing social ties through bonding activities … both in the early stages of the team development and the later stages, when social ties may fade and affect collaborative work” [Oshri 2008]. Further, some developers moved between onshore and offshore locations to get to know their partners better and to know more about their software development practices. This was seen in the case of Extol International, IONA Technologies, Telco, Manco, Consult, and BNP Paribas. Also, project leads played a very important role in co-ordinating development efforts and this was evidenced in three case studies. All these efforts were targeted to improving the interaction between the teams and could be seen to conform to the first value of the agile manifesto. Although several tools were used by the companies, it was clear that the importance given to people in the development effort was more than that given to tools and processes. The second value of the agile manifesto that stressed the importance of working software over documentation was also upheld in the case studies. Several companies instituted common build environments that were easily accessible to onshore and offshore teams. This indicated the current progress with the software and assigned responsibilities about the status of the software to certain developers or teams. Valtech India went to the extent of stating that “the golden rule of thumb was not to leave the office premises until the last code that was checked did not break the build” [Danait 2005]. As suggested by this value of the agile manifesto, documentation was kept at a minimum and the focus was on the working software. Documentation mostly included use cases, requirements, and burn-down charts. In case of Consult, it was reported that “often many documents were developed after the actual work was completed” [Balasubramaniam et al. 2006]. Among the principles of the agile manifesto, three received good support. Firstly, there was a strong emphasis in all case studies on communication.

Due to the distributed environment, the conversations became face-to-interface rather than face-to-face but it definitely took place on an as needed basis using a variety of electronic means. No exceptions were reported. Some companies like Valtech recorded kickoff meetings that took place via Web conferencing for the benefit of those who missed them and CEInformant even recorded discussion threads for subsequent reference if needed. Secondly, the need to deliver working software frequently was evidenced in nine case studies. The delivery period varied from two weeks (in case of SirsiDynix) to one month (in case of CEInformant). In the case of the Finnish company, the working software was shared between participants using a sprint demo and desktop sharing. This was a unique practice. Thirdly, the companies not only focused on delivering working software but they also measured progress in terms of the working software. SirsiDynix and BNP Paribas used the Jira software for doing so.

While two of the values and three of the principles of the agile manifesto were evidenced in the 12 case studies, some were prominently absent or at least there was no mention of any specific efforts made by the companies to uphold such values and principles. For example, the agile methods are widely touted to be customer focused because they take into account customers' late breaking requirements into the process of development and do not follow a plan. However, only one case study supported this value, eight did not say anything specific about it, and three were even against it. Two of the companies Manco and Consult went against this value from the very beginning and "devoted the first two or three iterations of a project to finalize critical requirements and develop a high-level architecture" [Balasubramaniam et al. 2006]. Some companies involved customers in the development process as suggested by the third value of the agile manifesto and the Finnish company followed a unique practice of involving a particular customer company's system architect as a proxy customer because they were planning to sell the software to multiple customers. However, most companies did not do anything specific to involve the customers in the development process and thus compromised the agile methodology to an extent. In terms of principles of the agile manifesto, some were poorly supported by the case studies. Since the case studies said very little about the involvement of the customers, it was not surprising that they did not say anything about whether any changes in the requirements coming from the customer at the later stages of the development were supported or not. Similarly, achieving simplicity in the design of the code was mentioned as a goal in only one case study and only two case studies mentioned that changes in team behavior took place to accommodate the needs of the process. Clearly, these principles were not given a lot of importance in the case studies. Project leads played an important role in coordinating development efforts and often acted as rotating gurus and visited offshore locations to liaise with the developers. Usually senior consultants were chosen for a responsible job like this. However, such a practice of building projects around qualified and motivated leaders was only illustrated in four case studies. Even more surprising was the finding that only five companies remarked that the offshore teams were self-organizing and enjoyed plenty of independence in developing the software. This basically showed that due to the distributed nature of the projects, it was important to bring in discipline to the process and so it was not always possible to give the offshore teams a lot of flexibility to do things in the ways they thought were the best. This finding was supported by literature on distributed agile development which reported that "for globally distributed projects, the conventional agile methods must be adjusted and modified by embracing more rigor and discipline in software development" [Lee et al. 2006]. Also, only five case studies showed support for the value of sustainable development and maintenance of constant pace of work. Often the developer had to put in long hours of work to deliver the working software and ensure customer satisfaction. It is interesting to note that, although the case studies showed strong emphasis on individuals and interactions, no specific steps seemed to have been taken by the companies to protect the developers from overwork and mental stress.

V. CONCLUSION

Increasingly companies are moving toward adoption of agile methods for distributed software development. Several challenges need to be overcome if the full benefits of agile methods are to be realized in a distributed setting. It must be remembered that not all projects can be handled using distributed agile methods. It is believed that for very complex and strategic projects distributed agile methods may not be a good choice [Moore and Barnett 2004]. In fact, most of the case studies that were studied in this paper were small in their scope. Also, it can be seen from the knowledge gathered from the 12 case studies that different solution strategies work for different companies based on available resources, intended outcome, and work culture. Several lessons were learnt from the case studies. Some of them were similar to lessons learned from any distributed software development projects whereas some were specific to the agile development. It was found that team selection, trust, and knowledge management were quite important for the agile case studies. The case studies were analyzed from the point of view of the agile manifesto and it was observed that some values and principles were adhered to in these case studies and some were either not followed or were compromised to an extent. Most case studies emphasized values like the important role played by the developers and the project leaders and their interactions and the need for regular delivery of working software often with minimal documentation. The principle of face-to-interface communication was also greatly championed by all case studies. A common build environment was commonly seen in the case studies that helped to measure progress of the project in terms of the working software and was shared between developers. The emphasis of agile methods on customer collaboration, responsiveness toward customers' changing requirements at any point of the development cycle, building projects around motivated individuals, emphasizing simplicity of design, creating self-organizing teams, and allowing flexibility and adjustment of team behavior according to need are some of the values and principles of the agile manifesto that were least discussed in the case studies. This indicated that some departure from the values and principles of the agile manifesto were needed. However, such a non-conformance is not surprising. There is evidence in past literature that the agile methods need to be carefully adjusted when applied to a distributed setting by embracing more rigor and discipline [Lee et al. 2006]. The same phenomenon was observed in the case studies that were studied in this paper. It must be noted at this point that the entries in Table 4 were based on author's interpretation of the 12 case studies and they made use of author's judgment. In future, a study can be done with more than one person filling out a similar table based on the case studies. The resulting tables can then be reconciled to find a more objective evaluation of the case studies and to determine to what extent the case studies followed the agile manifesto. The lessons learnt and the summarized knowledge from the case studies in this paper should be useful for practitioners and companies that are planning to venture in the world of distributed agile development and they will adopt these solutions and customize them according to their needs, strengths, and limitations.

ACKNOWLEDGEMENT

Thanks are due to the anonymous associate editor and the editor in chief for their useful suggestions on the earlier versions of the manuscript which has greatly improved the clarity, readability, and contribution of the paper.

REFERENCES

Editor's Note: The following reference list contains hyperlinks to World Wide Web pages. Readers who have the ability to access the Web directly from their word processor or are reading the paper on the Web, can gain direct access to these linked references. Readers are warned, however, that:

1. These links existed as of the date of publication but are not guaranteed to be working thereafter.
2. The contents of Web pages may change over time. Where version information is provided in the References, different versions may not contain the information or the conclusions referenced.
3. The author(s) of the Web pages, not AIS, is (are) responsible for the accuracy of their content.
4. The author(s) of this article, not AIS, is (are) responsible for the accuracy of the URL and version information.

Abrahamsson, P. et al. (2003). "New Directions on Agile Methods: A Comparative Analysis," in *Proceedings of the 25th International Conference on Software Engineering*, ACM Press, pp. 244-254.

Armour, P. G. (January 2007). "Agile … and Offshore," *Communications of the ACM* (50)1, pp. 13-16.

Austin, R. and D. Lee. (2003). "Beyond Requirements: Software Making as Art," *IEEE Software* (20)1, pp. 93-95.

Balasubramaniam, R. et al. (October 2006). "Can Distributed Software Development be Agile?" *Communications of the ACM* (49)10, pp. 41-46.

Boehm, B. and R. Turner. (September-October 2005). "Management Challenges to Implement Agile Processes in Traditional Development Organizations," *IEEE Software* (22)5, pp. 30-39.

Bowen, S. and F. Maurer. (2002). "Process Support and Knowledge Management for Virtual Teams Doing Agile Software Development" in *Proceedings of the 26th Annual International Computer Software and Applications Conference,* IEEE Computer Society Press, pp. 1118-1120.

Computer Enterprises. (2005). "Global Agile Development—CEI's Approach to Successful IT Outsourcing," http://www.ceiamerica.com/cei/company/resources/ceinformant_09132004.pdf (current July 30, 2007).

Danait, A. (2005). "Agile Offshore Techniques—A Case Study" in *Proceedings of the Agile Development Conference*, IEEE Press, pp. 214-217.

Fowler, M. (April 2004). "Using an Agile Software Process with Offshore Development," http://www.martinfowler.com/articles/agileOffshore.html (current March 26, 2008).

Fowler, M. and J. Highsmith. (August 2001). "The Agile Manifesto," http://www.ddj.com/architect/184414755 (current March 26, 2008).

Huo, M. et al. (2004). "Software Quality and Agile Methods" in *Proceedings of the 28th Annual International Computer Software and Applications Conference Vol. 1,* IEEE Computer Society Press, pp. 520-525.

Kussmaul, C., R. Jack, and B. Sponsler. (August 2004). "Outsourcing and Offshoring with Agility: A Case Study" in Zannier, C., H. Ergogmus, and L. Lindstrom (eds.) *Proceedings of the 4th Conference on Extreme Programming and Agile Methods*, Berlin, Germany: Springer, pp.147-154.

Lee, G., W. Delone, and J. A. Espinosa. (October 2006). “Ambidextrous Coping Strategies in Globally Distributed Software Development Projects,” *Communications of the ACM* (49)10, pp. 35-40.

Lindvall, M. et al. (December 2004). “Agile Software Development in Large Organizations,” *IEEE Computer* (37)12, pp. 26-34.

Massol, V. (2004). “Case Study: Distributed Agile Development,” http://www.pivolis.com/pdf/Distributed_Agile_V1.0.pdf (current November 30, 2007).

Moore, S. and L. Barnett. (September 2004). “Offshore Outsourcing and Agile Development,” *Forrester Research*.

Nerur, S., R. Mahapatra, and G. Mangalaraj. (May 2005). “Challenges of Migrating to Agile Methodologies,” *Communications of the ACM* (48)5, pp. 73-78.

Nisar, M. F. and T. Hameed. (2004). “Agile Methods Handling Offshore Software” in *Proceedings of the 8th International Multitopic Conference*, IEEE Press, pp. 417-422.

Oshri, I., J. Kotlarsky, and L. Willcocks. (April 2008). “Building Critical Social Ties for Global Collaborative Teamwork,” *Communications of the ACM* (51)4, pp. 76-81.

Paasivaara, M. and C. Lassenius. (October 2006). “Could Global Software Development Benefit from Agile Methods?” in *Proceedings of the IEEE International Conference on Global Software Engineering*, IEEE Computer Society Press, pp. 109-113.

Poole, C. J. (June 2004). “Distributed Product Development Using Extreme Programming” in Eckstein J. and H. Baumeister (eds.) *Proceedings of the 5th International Conference on Extreme Programming and Agile Processes in Software Engineering*, Berlin Germany: Springer, pp. 60-67.

Rees, D. (April 2004). “Distributed Agile Development,” http://www.itwales.com/998851.htm (current March 26, 2008).

Sepulveda, C. (2003). “Agile Development and Remote Teams: Learning to Love the Phone” in *Proceedings of the Agile Development Conference*, IEEE Computer Society Press, pp. 140-145.

Sutherland, J. et al. (January 2007). “Distributed SCRUM: Agile Project Management with Outsourced Development Teams” in *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, IEEE Computer Society Press, pp. 274-284.

Talby, D. et al. (July-August 2006). “Agile Software Testing in a Large-scale Project,” *IEEE Software*, pp. 30-37.

Yap, M. (July 2005). “Follow the Sun: Distributed Extreme Programming Development” in *Proceedings of the Agile Development Conference*, IEEE Press, pp. 218-224.

Yin, R. K. (2002). *Case Study Research Design and Methods*, *3rd Edition,* Thousand Oaks, CA: Sage Publications.

ABOUT THE AUTHOR

Indranil Bose is associate professor of Information Systems at the School of Business, The University of Hong Kong. His research interests are in the areas of telecommunication, data mining, software development, and supply chain management. He holds a BTech from the Indian Institute of Technology, MS from the University of Iowa, MS and PhD from Purdue University. His publications have appeared in *Communications of AIS, Communications of the ACM, Computers & Operations Research, Decision Support Systems, European Journal of Operational Research, Expert Systems with Applications, Information & Management, Journal of the American Society for Information Science and Technology, Journal of Organizational Computing and Electronic Commerce, Operations Research Letters* etc. His research has been supported by several grants from academia as well as industry. He is listed in *Marquis Who’s Who in the World, Marquis Who’s Who in Science and Engineering, Marquis Who’s Who in Asia,* and *Marquis Who’s Who of Emerging Leaders*. He acts as an associate editor of *Communications of AIS* and associate editor and editorial review board member for several other journals in the area of information systems

Communications of the Association for Information Systems

ISSN: 1529-3181