Analyzing the Applicability of an Agile Methodology to Distributed Collaborative Software Development

Tobias Hildenbrand and Michael Geisser and Denis Bruch

Working Paper 4/2007
January 2007

Working Papers in Information Systems

University of Mannheim
Department of Information Systems 1
D-68131 Mannheim/Germany
Phone +49 621 1811691, Fax +49 621 1811692
E-Mail: wifo1@uni-mannheim.de
Internet: `http://www.bwl.uni-mannheim.de/wifo1`

1 Introduction

Today, information technology (IT) has penetrated most domains of business and private life. The knitting of IT-systems and their dependencies are getting more complex every day. For businesses, this development can mean great opportunities. IT has become a main driver for competitive advantage and business success. On the other hand, misled software development (SD) projects can mean an existential threat to the operational and financial situation of a company. The efficient development of effective software is an essential part of optimally facing present and future challenges.

Managing SD with traditional methodologies often leads to high planning and management overhead and still, severe schedule deviations and budget overruns cannot be eliminated. The sequential and plan-driven traditional approaches are often not able to support an adequate reaction to either internally or externally caused changes in requirements. Complex and unclear system landscapes with diverse interfaces, ambiguous customer requirements, changing business strategies or fluctuating legal requirements are just a few examples for possible sources of changing system requirements.

In response to their experiences, in the early 1990s, different software developers introduced alternatives to the existing plan-driven and sequential approaches - the then so called "light weight" SD methodologies. In 2001, the seventeen leading researchers of this "agile movement" gathered to eventually write down their common values in the "Manifesto for Agile Software Development". These values addressed the importance of communication, working software, collaboration and handling of change. Agile development was the new approach of very experienced programming professionals to the challenges of complex software projects with vague and changing requirements. The group of developers mentioned above and today many others form the non-profit membership organization "Agile Alliance", which supports agile developers and maintains a large collection of publications on agile methodology.

Today, Extreme Programming (XP) is the most popular agile development methodology supported by the Agile Alliance. Its name was chosen because it claims to bring common sense to an extreme level (Beck, 1999b). It focuses on communication, simplicity, feedback and courage, to improve the speed and quality of SD. Formal processes and documentation are neglected in favor of tacit knowledge to improve flexibility. Close communication between developers and the continuous integration of customer representatives are key components of XP.

XP was initially developed for small to medium sized collocated development teams. This paper analyzes to what extent XP can be transferred to larger distributed developing endeavors. The focus is on XP, because it is the methodology with the highest congruence to the original Agile Manifesto. It does not claim to be all new, but to be an aligned composition of well established ideas and practices from other methodologies (Beck, 1999, p.73).

In section 2, agile SD is introduced in more detail, beginning with the general philosophy, continuing with an exemplary description of XP's principles and practices. Section 3 systematically analyzes the distributed application development scenario and its characteristics. Also, an overview over a selection of groupware supporting project communication is given. In section 4, the XP methodology is confronted with the characteristics of distributed application development and the suitability and effectiveness of available tools and techniques is discussed. Section 5 summarizes the results and gives a short conclusion.

2 Agile Software Development

Even though developed much earlier, public attention on agile methodology did not start arising before the late 1990s. Experience had shown that the plan-driven traditional approaches with their “heavy” sequential process models were not able to deliver successful projects. “The Chaos Report”, a study published by Standish Group in 1994, shows that merely 16.2 percent of their sample's SD projects could deliver on-time and in-budget. For the sample's larger companies the project success rate was not higher than 9 percent. Over half of the projects exceeded their initial budget by almost ninety percent and their time scope by 122 percent. 31.1 percent of the projects were canceled before completion. In average, the projects of the largest American companies achieved only forty-two percent of the originally planned features and functions. Software projects include a high risk, if new technology is explored or existing technology is pushed. The survey however shows that many failed projects dealt with established technologies, like in the cases of a drivers license database, a new accounting package and an entry system. Among the top-ranked success factors or reasons for difficulties and failures, respectively, were the quality and stability of requirements, the quality of plans and estimations, and the degree of user involvement (The Standish Group, 1994).

Agile SD aims at exactly these deficiencies. Small, functional increments are iteratively developed to gain fast customer feedback and timely adjustment of requirements. The utilization of tacit knowledge through communication replaces extensive documentation and makes development more flexible. The following subsection introduces the agile philosophy on the basis of the Agile Manifesto, before XP's practices and principles are presented.

2.1 The Agile Philosophy

Even though distinct in their emphasis of the different key issues, a series of agile or lightweight development approaches existed in the late 1990s. In early 2001 the intellectual leaders of this agile movement, the creators of the different development methodologies met with the intention to find out what their methods had in common. They developed the following "Manifesto for Agile Software Development":

We are uncovering better ways of developing software by doing it and
helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

(Beck et al., 2001)

The first value addresses the potential wasted, when intelligent people, as programmers mostly are, are restricted and overloaded by processes and a given set of approved tools. Direct communication between involved individuals and the utilization of tacit knowledge are understood as the most efficient and effective ways of error detection, ad-hoc coordination and product improvement (Highsmith and Cockburn, 2001, p.121). In contrast, high levels of strict process compliance are both, perceived as restraining creativity and generating avoidable overhead. The agile philosophy aims at providing developers with a supportive working environment and motivating them by allowing to work widely self-determined and self-organized.

The second value approaches the traditional perception that every development step must be thoroughly documented. In contrast, agile methods replace extensive documentation by producing simple code in small increments, frequently tested and improved. Extensive documentation is considered as "shelfware", mostly outdated before bound (Boehm and Turner, 2004). Small increments of functioning software are considered as unforgiving honest (Highsmith and Cockburn, 2001, p.121) and the best way to win the trust and confidence of customers.

The third value emphasizes the importance of customer collaboration and integration. Software, if produced on demand, is very customer-specific, hence project success is strongly dependent on the collaboration of knowledgeable customer representatives (Boehm, 2002, p.66). Client-side business representatives are meant to drive the evolution of requirements and provide the development team with frequent feedback about delivered increments. Customer feedback indicating needed change is not regarded importunate, but as essential for the achievement of the overall project mission – customer satisfaction. Extensive up-front contract negotiations in contrast are perceived as baffling because requirements, which are most likely to change, are designed and fixed before they are actually well-established.

This leads to the last value of responding to change. Under agile aspects it is not seen as sufficient to satisfy customer needs only from the analysis to the requirements definition phase, before implementing software irrespective of occurring changes. Agile approaches aim at continuous customer satisfaction throughout the whole development life cycle. Changes in requirements, scope or technology are uncontrollable to the development team, hence the only viable strategy is to keep the costs of change as low as possible (Highsmith and Cockburn, 2001, p.120). Documentation, extensive up-front planning, and complex “future proof” architectures lead to high costs of change, because complex structures have to be refactored. Agile methodologies promote lean documentation, incremental and iterative development, and simple design. This way the architecture is kept flexible while outdated documentation and code are minimized. This responsiveness and flexibility in combination with minimal documentation creates new challenges in terms of traceability of processes and changes. However, traceability is required by many industry standards, e.g. by the FDA for software in medical engineering.

2.2 Extreme Programming

XP was first developed in a Chrysler project in 1975, simply as means to get the job done (Beck, 1999, p.75). Today it is the agile methodology receiving the most public attention (Rumpe and Scholz, 2002) and possessing the highest publication coverage. So far, it has been successfully implied in many projects (Martin and Schwaber, 2004; Karlström, 2002).

Beck (1999b) identifies the customer, programmer, tester, tracker, coach, consultant and manager as the seven roles in XP. Since agile methodology is very people centric, these shall be introduced first. The most important roles are those of the customer and the programmer (Martin, 2000, p. 12). XP demands to permanently integrate a **customer** representative in the development team. The customer's job is to identify the necessary features (called “stories” in XP), prioritize the stories and design functional acceptance tests to determine whether a story is successfully implemented. The **programmers** estimate the effort the different stories will take and implement them according to the customer's prioritization. They write the unit tests and keep the code as simple as possible. Programming in XP is understood as a permanent dialog between a programmer, his pair programming partner and the customer (Martin, 2000). The **tester** supports the customer in writing functional tests and is responsible for the overall maintenance of the testing tools. The **tracker** traces project progress and gives feedback on the accuracy of estimations. During iterations he evaluates progress and identifies possible changes to be made. The **coach** is an XP expert, who guides the team to a correct conversion of methodology to practice. **Consultants** are external technical specialist, integrated to help the team solve specific technical issues. The **manager** is the one, who has the strategic project responsibility and makes the final decisions. XP is very sensitive to the behavior of each and every team member. Every one has to be convinced of the approach and participate in order to make the project successful (Beck, 1999, p74; Abrahamsson et al., 2002, pp.21-22).

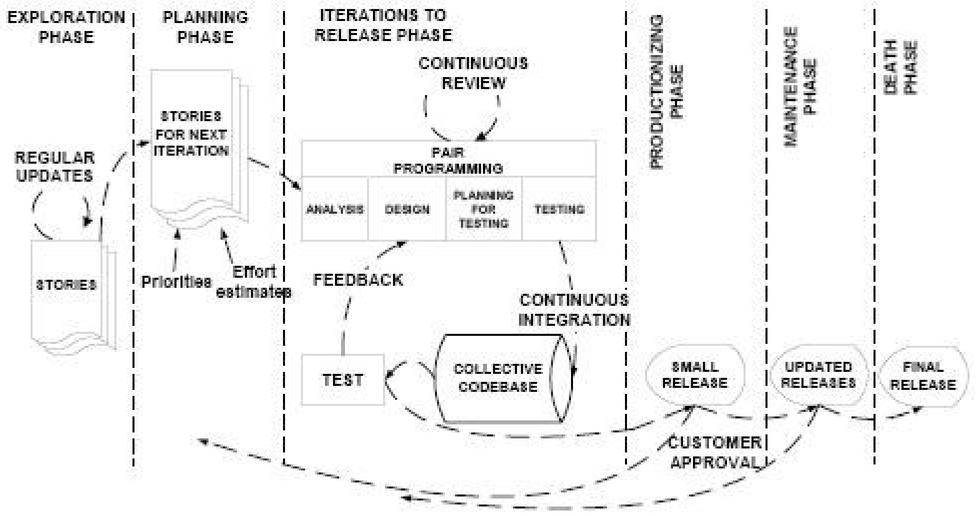

Figure 1: The XP Development Life Cycle (Beck, 1999b)

The development cycle of XP consists of six phases (see *Figure 1*). Projects begin with the **exploration phase**. In this phase the development team gets familiar with its development environment and the technology it will be addressing. The technology and the architecture of the system are explored. Meanwhile, the customers write the stories to be implemented (Abrahamsson et al., 2000, p.20). The collection of stories to be implemented is maintained and expanded throughout the development life cycle. In the **planning phase** customers assign priorities to their stories and developers estimate the necessary effort for their implementation. Then a set of stories for the first small release is agreed upon and the release is scheduled according to the programmers estimations. In the **iterations-to-release phase** the actual implementation is done. For each iteration the customer chooses the smallest set of most valuable stories that make sense together (Beck, 1999, p.72) and programmers produce the functionality. Code is always written by pairs of programmers to improve quality and increase development speed. A release consists of several iterations each lasting from one week to one month. Iterations are divided in several tasks programmers can take responsibility for. Tasks are split up in small test cases. XP coding always begins with the development of unit tests. After the tests are written the code is developed. The code is then continuously integrated and tested. All tests must be passed before the developer moves on to the next test case. During this time the customer specifies functional tests. At the end of the iteration these functional tests should all be running, before the team can continue with the next iteration (Beck, 1999, p.72). After all iterations scheduled for the release are completed the system is ready for production. The **productionizing phase** consists of extra testing of functionality and performance. Changes which might still be necessary are assessed and scheduled. Either they are implemented in short iterations or postponed to the maintenance phase or later releases. The phase ends with a finished release, which is delivered to the customer. During the **maintenance phase** the system must be kept running in production, while remaining stories are implemented in further iterations. The velocity of development can be decelerated and the team may be restructured, depending on the amount of remaining stories. Development stays in this phase until the system satisfies the customers' needs in all aspects. Finally, development enters the **death phase**. Now that no changes to architecture, design, or code will be made any more, documentation is finally written. The death phase could also occur any time earlier, if the customer does not expect to receive any further desired functionality at justifiable expense and cancels the project. In that case, the customer remains with the functioning software developed up to that point (Abrahamsson et al., 2000, p.20).

XP project participants are urged to live the four XP values derived from the agile manifesto: communication, simplicity, feedback and courage. **Communication** is perceived as important means to retrieve tacit knowledge, concerning the customer's requirements, the colleagues' troubles with implementation or any other relevant issue. **Simplicity**, the reduction to the necessary, is the core value to achieve speed and agility. **Feedback** of the customer helps identify misled or insufficient development at the earliest time possible. **Courage** expects people to try new directions and mine new opportunities. The risk of going a wrong direction is severely reduced by communication and feedback (Beck, 1999).

Thirteen core XP practices make up its philosophy (Beck, 1999, p.71): In the **planning game** programmers estimate the effort necessary for the existing stories. Stories are written on story cards and together developers and customers shift around cards to find a combination that makes sense for the next release. The customer decides about the scope and duration of the next release by making the final selection of stories to implement. **Small releases** reduce the risk of misled development and help develop the customer's trust and confidence in the development team. This way, the scope of the releases stays manageable without extensive documentation. Customers and programmers share a **metaphor** or set of metaphors describing the system. This aims at a better mutual understanding (Abrahamsson et al., 2002, p.24). Programmers keep the **design simple**, by reducing it to the necessary minimum. The designed program may not contain any unnecessary features or duplicate code and must have the fewest classes and methods possible and still pass all tests. This way code stays clear and the costs of change are kept as low as possible. Unit **tests** are written by programmers before the code is developed. They are integrated in a test suite, with which code is continuously tested. This way errors are detected very quickly and the quality of code is kept high. Simultaneously, the customer writes the functional acceptance tests. Since code and design evolve throughout the development life cycle, a regular **refactoring** of the code is necessary for the consistency of design. The complete test coverage through the integrated unit tests, makes refactoring safe, because the proper function can permanently be checked. The running test suite guaranties the validity of the refactored code. As mentioned before, code is always **pair programmed**. Two developers on one computer can keep the quality of code high, because they always have to convince their partner of the superiority of their solution and they can mutually find their mistakes. In pair programming, knowledge is passed from one to the other very quickly, almost osmotic. New team members can be paired up with experienced ones, this way grabbing on to the project extremely quick (Cao et al., 2004, p.5). **Continuously integrating** programmed code, which means several times a day, urges to keep development units (test cases) small and maintain a high development pace. Integration leads to a new system built and to quick error detection. **Collective ownership** of code empowers everyone, who finds an opportunity of improvement to change the code accordingly. Again, the test suite's coverage of the code guaranties the validity of changes. An extremely important part of XP is the permanent availability of the **on-site customer**. He represents the users of the system to the development team designing tests and helping to resolve obscurities. The collaboration of a knowledgeable, representative, and committed customer is a key success factor of XP projects (Boehm, 2002, p.66). To maintain a sustainable pace, developers should not exceed a **40-hour week** twice in a row. XP classifies continuous overtime as a problem, which needs to be solved. (Beck, 1999, p.71) Overstraining developers leads to low quality of work and low programmer motivation and contentment. Emphasizing the power of direct communication between developers, XP demands an **open workspace** to facilitate communication throughout the team. Informal communication is understood as the fastest and most effective way of spreading tacit knowledge. Individuals working on a solution to a problem are understood to be most efficient when discussing the problem in direct face to face dialogs. XP team members have to commit to a number of rules and conventions. Following these rules is essential for the success of the whole team, but as they are **just rules,** the team can change these rules at any time. Before, team members have to agree on how they assess effects of the changed rules (Beck, 1999, p.71). This reflects that XP is agile in itself and the support of developers and not process compliance is the center of interest.

XP can also be understood as development through constant dialog. Developers communicate amongst each other to efficiently utilize tacit knowledge and quickly find new solutions to current challenges. Developers communicate with customer representatives to deliver the most valued features, gain rapid feedback on deliveries and improve the customers' trust and confidence. (Martin, 2000, pp.12-13)

3 Distributed Software Development

SD generally demands a high degree of collaboration between the involved experts as software developers spend large parts of their time working with others (de Souza et al., 2002). As size and complexity of development teams rise, the need of efficient communication and coordination processes grows. When SD projects are distributed over different buildings, countries, or even continents, communication is aggravated by spacial, temporal, and cultural differences. The following subsection shows different reasons and constellations for the distribution of projects before different consequences and challenges of distribution are discussed. The final part of this section addresses groupware technology to support communication of distributed development teams.

3.1 Distribution of Development Teams

Reasons for the distribution of development can be divided into three major categories according to the parties involved. Collaboration can be in a company-internal context or a company external context. *Company-external collaboration* is either between cooperating development firms or between development firms and a client company. Especially in context of agile methodologies the sustainable and extensive cooperation between customers and the development team takes an important role. *Company-internal collaboration* involves project units, distributed over different locations. This may be, because of the historical development of company branches, the organization of technological clusters in certain regions, or due to reduced costs of labor in different countries, for instance in Eastern Europe or Asia.

Collaboration between different development companies involve different parties of specialists working on one common goal. These can be projects of equal cooperations with mutual responsibilities, but also sub-projects given to an outsourcing partner. In SD projects, domain experts are often hired externally and included in a project team. These are often highly mobile and can only be integrated remotely.

The collaboration with customers is highly important for SD. Traditional approaches have business analysts collaborate on-site with the customer during the analysis and requirements engineering phase whereas XP demands a continuous integration of highly knowledgeable customer representatives during the whole development life cycle. Customer companies are very unlikely to abandon their most knowledgeable staff members to exclusively work off-site with a development team. In this case, parts of the development team have to be on-site with the customer and need to be remotely integrated with the rest of the team, or the customer representatives themselves are remotely integrated in the development team.

3.2 Consequences and Challenges of Distribution

Teams working within one office building see each other daily and are aware of the activities others currently encounter. Projects include regular meetings to evaluate project status, resolve issues, and plan further steps. When problems arise ad-hoc communication or stand-up meetings can be arranged to find an instant solution. By regular interactions team members get familiar with each other and can assess the reactions of their colleagues in certain situations. This leads to personal relationships with an enhanced mutual understanding and team spirit. This paper identifies three different issues arising with distribution – spacial distance, temporal and cultural differences.

The *spacial separation* leads to the exacerbation of communication and coordination. Generally, direct communication is replaced by computer-mediated communication. Modern technologies deliver several ways to globally communicate with integrated video and application sharing functionality. Still, there are a few factors, which cannot be compensated: Social aspects can make a large difference between medial and direct communication. The relationships built in virtual working environments cannot be compared to those in face to face meetings. Additionally, verbal communication is enhanced by body gestures and face expressions. These are hard to be assessed accurately in computer-mediated meetings. Depending on the degree of distribution, overcoming these restrictions with real life meetings can be of course very time and money consuming.

Teams developing software in globally-distributed scenarios also have to cope with different working hours due to the *different time zones* they are located in. This can lead to enormously long response times and restricts synchronous communication to the intersection of working hours. Developers in Germany working with developers in India have a time difference of 3.5 hours. This might not seem much, but can mean that developers in India have to wait over three hours until their colleges in Germany arrive in office to collaboratively resolve an urgent issue. On the other hand, developers in Germany have to wait for the next day until they get their issues resolved that arise in the last three hours of their working day. In some situations, working times may be slightly adjusted to minimize this effect, but if developers from further regions are included this will not be sufficient. The impact of time differences on collaborative projects between the USA and India, for instance, leave only very few hours where project members are in office on both sides (Simons, 2002). Asynchronous work can lead to impediments for SD projects, especially when development is expected to proceed rapidly.

The issue of *cultural differences* does not directly result from distribution, but it is most likely to accompany many distributed scenarios. Different criteria of culture can be identified. The most obvious cultural characteristic is the language. Language being the most important means of communication makes common project language almost indispensable. But even the fact that every project member speaks the project language, does not mean that everyone masters it to the same degree. Idioms, acronyms, and slang can be serious impediments to mutual understanding. Another aspect leading to misunderstandings and complicating communication are cultural customs. For instance, people from India do not use the word “no” to negate a question. Being very polite, they use complex descriptions which can lead people, who are not familiar with this cultural feature, to severely wrong conclusions. Beside these communicational issues, Hofstede (1994; 2001) identifies five different dimension of culture: *power distance*, *individualism*, *masculinity*, *aversion toward uncertainty* and *long vs. short term orientation*. **Power distance** is the extent to which the less powerful expect and accept that power is distributed unequally. **Individualism** describes the degree to which people define themselves as independent. **Masculinity** explains the degree of male attitudes like determination and competitiveness vs. female attributes like care and humility found in a culture. The **aversion towards uncertainty** is the degree to which someone is scared by uncertainty and ambiguity (Hofstede, 1994). Finally **long vs. short term orientation** describes whether people of a certain culture rather approach long term targets or are more likely to aim at momentary achievements (Hofstede, 2001). These cultural dimensions can have severe influence on working habits and organizational practices, which is of special interest in the context of agile methodology. The consideration of Hofstede's cultural dimensions goes beyond the scope of this work and must be postponed to further analysis. Table 1 summarizes the introduced characteristics of distributed development scenarios and matches aspects to be considered accordingly.

Characteristic	***Aspects***
Spacial distance	Indirect communication -> effectiveness of communication & social aspects
Temporal difference	Asynchronous working hours -> less real time communication & complex coordination
Cultural differences	Communicational & behavioral issues -> misunderstandings & exacerbated communication -> suitability of management approaches

Table 1: Characteristics of Distributed Development Scenarios

3.3 Communication and Groupware Technology

Groupware or computer supported cooperative work (CSCW) systems refer to technology supporting groups simultaneously working on a common goal. The scientific field of CSCW is highly interdisciplinary and addresses IT and information systems, sociology, psychology, managerial and organizational disciplines as well as several other scientific fields. For this work, the technological features for the support of integrating globally-distributed group members and their sufficiency in supporting agile principles are of main concern.

CSCW systems equip the group members with means of communication and interfaces to a shared working environment. This way, group awareness, the knowledge about the current activities of other group members, is fostered even in highly-distributed settings. Teams are empowered to coordinate their work processes more efficiently. CSCW systems aim at faster information transfer, better utilization of tacit knowledge, speeding up development processes and reducing administrative overhead (Schlichter, 2002, pp.11-15). These goals are remarkably symmetrical to the ones of XP's agile methodology.

The space/time matrix model according to Grudin (1994) classifies CSCW systems according to their support for different degrees of spacial and temporal distribution. Table 2 shows the dimensions with their distinction of same, predictably different and unpredictably different spacial and temporal location, with examples of suitable groupware options.

Time --->

Space --->	Same	Different and predictable	Different and unpredictable
Same	Meeting Facilitation	Working Shifts	Team Rooms
Different and predictable	*Video-/Tele-Conferencing*	*Electronic mail*	Collaborative Writing
Different and unpredictable	*Interactive Multicast Seminars*	*Computer Boards*	Workflow

Table 2: Groupware Matrix for Distributed Collaboration (Grudin 1994)

In the context of agile methodologies and distributed SD, the four sectors resulting from different spacial and identical or different but predictable temporal location are of relevance to our analysis. Adequate communication tools for corresponding scenarios are introduced in the following. Focus is on tools enhancing communication, as agile methodology is very communication intensive and aims at the minimization of process tool dependency.

Messaging Systems

Two kinds of messaging systems can be distinguished, asynchronous and synchronous messaging systems. Asynchronous messaging, in the form of e-mails, is a well established medium for coordination and the exchange of information over the Internet. Via mailing lists members of collaborating groups can keep each other up to date, synchronize and coordinate their work. Depending on the available bandwidth, any electronic data can be comfortably attached and distributed. With digital signatures and certificates, email can be securely used even for the exchange of confidential business information.

Synchronous messaging is often realized with instant messengers (IM). Participants are equipped with a client application providing a user interface to conveniently edit and read messages as well as send and receive files. Communication over IMs happens in real-time. A message sent by one participant is displayed in the user interfaces of all designated recipients that are currently online. Clients enable the user to set the own availability status and view that of their messaging contacts. This way, IM clients enable something like a shared virtual office, reducing the need of scheduled appointments. IMs are made publicly available by different providers or are included in collaborative software suites. Some clients also support audio and video functionality. With the necessary bandwidth and processing power available, low budget headsets and cameras can enhance messaging systems to audio, or even simple video conferencing systems.

Audio-Conferencing Systems

Today, audio conferences can be realized over ISDN (Integrated Services Digital Network) telephone lines, an Internet based VoIP (Voice over Internet Protocol), or mobile communication standards. Spoken communication is much more efficient for the clarification of complex or difficult matters. Being able to hear the opponents' voices enhances the exchange of information tremendously, especially in emotional regards. Where messaging systems only leave the possibility of expressing emotions with so-called “emoticons” (smileys etc.), audio conferences are able to transfer genuine emotions via voice. An effective utilization of audio conferences requires a bit of experience and discipline. Uncoordinated discussions are likely to end in inefficient chaos, especially with a large number of participants. Audio conferences enable synchronous communication, but therefore are usually less spontaneous and require scheduling.

Video-Conferencing Systems

Video conferences extend audio conferences by video signals, which are recorded on every site and shared with all other sites. This way the closest simulation of collocated meetings is achieved. All participants are able to see each other, hence the interpretation of face expressions and body gestures becomes possible. Video conferences can move distributed teams closer together. The transmission of factual content but also of emotions is effectively enhanced. Like audio conferences, video conferences usually require some up-front planning and scheduling.

Video conferencing equipment is rather expensive and must be properly set up in order to achieve optimal results. Like audio-conferencing, video conferencing can be realized over ISDN lines as well as over the Internet. Small groups can receive a deterministic quality over ISDN lines. If multiple participants are involved, the importance of broad Internet connections grows vastly. The quality of the video signal and its display should be of high quality, as low quality does not achieve the added value desired. In the opposite, bad video quality basically leads to distraction and annoyances.

New mobile communication standards also make mobile video conferencing possible. Video quality is very limited due to the compact equipment integrated in cellular telephones. The effects reached are not comparable to those with stationary video conferencing equipment.

Wikis

A wiki is a website, which can be edited in the browser itself without requiring the user to have advanced web editing skills. Wiki systems include a change tracking functionality, so that any previous state of a page can be restored. It is an easy way of collaboratively editing documents. An extremely high degree of linkages between different entries and file attachments is characteristic for wikis. Wiki systems usually offer a complementary discussion forum with threads pertaining to the different entries. These threads can be used by editors to discuss and comment changes. Changes can be propagated automatically by email.

Screen Sharing Software

Screen sharing applications enable participants in distant locations to simultaneously display identical screen content, according to the principle of "what you see is what I see" (WYSIWIS). The host of a session can edit and navigate through the viewed documents. Screen sharing tools or functionality are a very good way to enhance the efficiency of video and audio conferences.

Group Editing Software

Collaboration requires different people to contribute to shared documents. Group editing software enables group members at distant locations to simultaneously work on the same documents. Consistency of concurrent changes is maintained by sophisticated write protection mechanisms. Group editing software enables the collaborate creation of work products in video and audio conferences.

Continuous Integration Tools

Continuous integration tools monitor the versioning system for changes and automatically rebuild the software after changes are made. The responsible programmers are automatically notified about the successes or failure of the build. Tools like CruiseControl (cruisecontrol. sourceforge.net) generate reporting web pages, which enable all team members to monitor the status and current changes of the developed code. With a globally common code base, team awareness is supported, by keeping developers in distributed projects informed about the overall system status. Central code repositories and remote building processes require fast and reliable data linkages. (Fowler, 2004, p. 3)

Issue Tracking Systems

In development projects, various issues arise, which are complex and cannot be solved instantly. Issue tracking systems are supposed to manage these issues. They maintain a list of currently unresolved issues with description, comments and status. People working on an issue can post comments and edit the issue status, which indicates the priority, the degree of completion, and the current processor. When the state of a task changes all stakeholders are notified. Issue tracking systems help to coordinate tasks amongst project team members and serve as a knowledge base. Resolved issues should always be documented to facilitate quick resolution of similar problems.

4 Applicability of Agile Methodology to Distributed Scenarios

Physical distribution of project team members among different development locations makes some of XP's practices less feasible. The open workspace, facilitating direct communication, and enhancing the team awareness, can only be realized in relatively small collocated teams. The practice of pair programming, in its original form, demands two developers physically sharing one set of computer hardware. XP's strong integration of customers demands the constant availability of an on-site customer. This is only feasible, if the distance of the customer's location to the development team is rather small. Since the distance to the main development site can be very large, the local presence of a customer representative is often not viable. Other practices of XP are very beneficial for distributed development projects: Continuous integration, if practiced with discipline, can eliminate major integration issues common to distributed development scenarios. Team communication and customer integration are the XP domains, which are affected strongest by distribution and are therefore discussed separately in following subsections. Other XP development practices concerning distributed scenarios are examined subsequently.

Similar to Wallace et al. (2002), this paper will distinguish remote project participants in near-site and off-site customers and team members, respectively. Near-site indicating that the workplace is not shared, but distance can be traveled with only little effort. Stakeholders being off-site, are geographically-located very far away, work in a different time zone and might have a cultural background different to that of other project stakeholders.

4.1 Team Communication

Software developers spend a very large part of their time working with others (de Souza et al., 2002, p.1). The essential role direct communication takes in XP leads to additional project risks when development teams are physically distributed (Kontio et al., 2004, p.1). Open communication requires trustful relationships. Collocated teams spend much face time working or informally coming across each other in cafeterias, hallways or elevators. They constantly communicate business matters and also share some private issues. Valuable personal relationships and foundations for collaboration are built and maintained in a way that is not viable in distributed scenarios. The communication technologies introduced earlier can be very profitable, but for the development and maintenance of trustful relationships face time is essential.

For off-site XP development scenarios, Fowler (2004) suggests mutual seeding and maintaining visits between the distant locations to build and maintain relationships as a basis for efficient communication and collaboration. Seeding visits should be scheduled early in the project. They are to be connected to some joint tasks, to get the team members used to working together. The workload should allow a relaxed work pace to still leave enough time for the creation of valuable personal relationships. The visits should be scheduled for a sustainable duration to really get people to connect. The shorter maintenance visits, scheduled later during the project, serve to maintain and intensify relationships (Fowler, 2004, p.4).

For the main part of the development life cycle the majority of developers are located at their home development sites. Similar to the communication responsible moderators of large XP development teams (Rumpe & Scholz, 2002), distributed XP development teams can be enriched by a moderation role, responsible for communication between the development sites. The site-moderation role should be taken by communicative team members with good standings inside the team. Ideally the moderators of the different sites have personally met and established a trustful communication basis with the other sites. It is the site-moderator's responsibility to initiate information exchange wherever necessary and to ensure that the team members' communication barriers are held low. Every day, at fixed times, moderators discuss current issues and project status in audio or video conferences. Especially in scenarios with severe time differences moderators should prepare conferences to efficiently deal with all current matters, because direct communication is reduced to very few occasions. To avoid annoyances, both sides should sacrifice some early or late hours for the held conferences (Fowler, 2004). Moderators should include other team members in their meetings when necessary. Intra-team meetings are arranged regularly to achieve a broader exchange of current issues and foster the team awareness. The moderation role should be rotated through the developing team regularly. This way communication is not too strongly influenced by a single person, a wider basis of inter-site contacts is established and moderators do not loose touch to the actual development tasks.

To enhance distributed project teams with important cross-locatio contacts, development sites can exchange ambassadors. Ambassadors have many good contacts to their home-site and thus enormously improve communication in both directions. Sent to off-site locations, ambassadors can improve the understanding of business context and of cultural differences. Ambassadors facilitate the exchange of informal and tacit knowledge (Fowler, 2004, p. 3).

The introduced measures are important for off-site as well as near-site scenarios. In near-site development collaboration, the team spirit and team awareness related issues are easily neglected because of the physical proximity. The face time between developers, achieved when near-site exchanges are arranged, drastically exceeds the face time that is caused through joint meetings. Relationships and insights are improved by exchanges in near-site scenarios, as they are in off-site scenarios. Efficient collaboration can be tremendously improved, if the possible roots of communication deficiencies are considered early on.

Communication between distant development sites is based on the concepts of mutual visits, communication responsible moderators and mediating ambassadors are adapted sufficiently. Contacts are initially established by seeding visits and fostered by maintaining visits and ambassador exchanges. The moderator's role, as a communication facilitator, ensures constant information flow between sites.

The open workspace required for the XP methodology is not imitated sufficiently by the precedent measures alone. Team members should be able to easily access information and knowledge at all sites. In distributed scenarios this is best-enabled by the introduced groupware technologies. The open and trustful relationships lower communication barriers and leverage communication efficiency.

For teams working in time zones with a sufficient intersection of working hours, IMs are a very good basis for close collaboration. If availability information is maintained consequently, developers can approach their distant colleges ad hoc and get very quick responses. Short response times are extremely important in the context of XP's rapid development pace. When issues cannot be resolved efficiently through messaging, integrated VoIP telephony function or a regular telephone can be used. Telephone conversations can be enhanced by the sharing of files through an IM or via email. For documents requiring mutual input group editing software can be employed. IM conversations, telephone conferences and group editing sessions can always incorporate multiple participants. This way, even the spontaneous stand up meetings, which are characteristic to XP, can be arranged to solve current issues. The availability of video conferencing systems can enhance discussions and makes them more personal. Especially for people, who are not very familiar with one another, video conferencing can leverage the effectiveness of communication processes tremendously. Casual meetings in hallways, kitchens or elevators can be an important source of information (Schlichter, 2002, p. 23). Different experimental projects at Xerox, Accenture or Microsoft Research have set up permanent video-conference linkages between different rooms in distributed environments, to enable the important informal exchange of information (Braun et al., 2001, p 13). This could be an effective simulation of collocation, leading to efficient osmosis of information and better personal contacts between distant team members.

Teams developing in time zones with very little or no intersection of working hours, as for instance the USA and India, there are hardly any occasions for synchronous communication. Severe time differences have severe impact on agile methodology. Less synchronous communication makes the facilitation of relationships and connections between development sites even more vital. If locations are not assigned to distinct development domains, the process of handing over work becomes essential. This demands better tool support, documentation and discipline. If questions arise, which can only be answered by the off-site team, development can be delayed for many hours. If dialogs have to go back and forth more than once, short iteration schedules are destroyed very easily (Simons, 2002, p.3). Wikis can provide space to publish certain basic rules and conventions to enable better collaboration. A continuous integration tool with a common code repository, can hand over the development from one team to another, by providing developers with reports about all changes made. The utilization of issue tracking systems can provide support for the coordination of tasks as well as short term documentation of current issues. The XP value of communication is severely influenced by large time differences.

The technical realization of pair programming can be done with constant audio and video connections combined with group editing functionality. To enable reasonable collaboration, the required speed and quality of the connection is very high, especially if several teams are paired between the distant locations. The permanent use of a headset can be very exhausting for the developer and reduces the connectivity to collocated team members, hence pair programming generally favors collocated developers. In individual cases where developers from different locations are optimally suitable to solve a special problem together, distributed pair programming can be good option. For the transfer of knowledge and skills between developers from different sites, visits and exchanges should be utilized to pair up programmers. Pair programming is a practice not suitable for permanent application between developers in distributed locations. For locations with large time differences distributed pair programming is generally unfeasible.

4.2 Customer Integration

Integrating a customer representative in the development team is a central practice of the XP methodology. The on-site customer develops stories, provides feedback to developers, and creates acceptance tests. The availability of collaborative, knowledgeable customers is generally difficult, as those are the employees client companies usually do not want to spare. In distributed development projects, this difficulty is accompanied by additional challenges. Large parts of the development is done off-shore or development teams are distributed among several locations. Both cases make it unreasonable for a customer to permanently attend all development sites. Integrating the customer remotely can improve the quality of the customer representatives appointed by the client company. Remote integration and surrogate customers can make the XP approach much more comfortable for the client company.

The XP methodology integrates customers closely in the development team, hence communication issues are very similar to those discussed in the previous subsection. Communication is more effective and provides more valuable information, if it is based on a solid relationship. The reduction of communication barriers towards the customer is extremely important. The relationship can only work, if both sides are committed (Wallace et al., 2002). The introduced moderators or another team member should take special responsibility for the maintenance of an effective flow of information to and from the customer. If a face-to-face meeting between the customer and the development team is not feasible, a video conference should introduce the involved customer representatives and the whole development team early in the project to lower initial communication barriers. With large time differences this can be inconvenient for development teams off-shore, but should be treated openly and considered as a possibility to emphasize the international character of the project and the involved companies.

In the planning game the customer decides about the scope and duration of the following release. This process is very important to the course of the whole project and for the achievable customer satisfaction. If possible, near-site customers should do this with programmers in face-to-face meetings (Wallace et al., 2002, p.135). Since this step is decisive for the project success, a working trip to the customers' site should also be considered. Otherwise, off-site customers should get involved with the programming team as closely as possible by mobilizing all technical means available. A portion of extra time should be calculated for sufficient release planning activities with distributed customers.

During the iterations-to-release phase, the overall tasks should be defined and customer collaboration mainly consists of writing the acceptance tests and providing quick feedback (Wallace et al., 2002, p.136). Due to the close collaboration between testers and customers during the development of functional acceptance tests, the testers role gains additional importance. During the iteration-to-release phase the tester's role is capable of additionally functioning as a communication enhancer. The knowledge mined during the development of tests can be of great use and must be shared with the whole team. Group editing software combined with IMs and audio or video conferencing, are an adequate groupware configuration for the remote development of functional tests. While customers and testers are working on acceptance tests, programmers do the SD. To improve the customers' reaction times on feedback requests are essential. Due to the narrow scope of small increments, development progress can be severely delayed if response times are too long. In addition to speed, the quality of information exchange is highly important to get issues resolved correctly in the first approach (Simons, 2002, p.3). Customer representatives should be equipped with an IM to be able to answer arising questions as quickly as possible. Timely responses of customers to development-related e-mails can help development teams in distant locations fulfill time critical tasks. A strict prioritization system should be established to help the customer rate the urgency of requests. The customer must be made aware of his role in XP to achieve the necessary project commitment, which especially vital in distributed scenarios.

Whether customer representatives are integrated locally or remotely, the involvement in XP projects heavily impacts their normal scope of responsibilities and working habits. An alternative or supplement suitable to XP, is the assignment of business analysts as surrogate customers (Cao et al., 2004, p.5). Surrogate customers represent the programmers' interface to the client company. It is their job to analyze clients' business needs. As suggested by Wallace et al. (2002, pp.135-136) for projects with many customers, the surrogate customer's assignment can span from iteration support to nearly the complete replacement of direct customer involvement. Surrogate customers tremendously reduce the teams efforts of maintaining outside relationships. In how far this can effectively imitate real customer integration, requires extra research.

4.3 XP Development Practices

Some of XP's practices can be very beneficial to the success in distributed development scenarios. In the following, these practices are examined.

Continuous Integration

In distributed projects, the segmentation of development tasks enables teams to develop autonomously, facilitating their daily business, until the day that the developed parts are integrated. Integration of separately developed application segments often leads to huge amounts of unexpected rework. Continuous integration urges developers to integrate the developed code several times daily. The continuous integration of small bits only leads to small and manageable amounts of rework. A central repository combined with an automated integration tool is a big advantage when it comes to accurate scheduling and the prevention of big integration disasters (Simons, 2002, p.2). Additionally automated integration tools can generate reports of recent changes, supporting the hand-over of work between distant development sites, with hardly any possibilities of synchronous communication (Fowler, 2004, p.2).

Small Releases

Distributed projects feature decreased visibility of the project's status for management and customers. Plan deviations and false estimations are often discovered very late, leading to unnecessary costs and quarrels. Developing software iteratively, with constant delivery of small functioning increments, allows project stakeholders to get valuable insights into the actual project progress. (Simons, 2002, p.2) Trust and confidence in the development team, as well as the stakeholders' commitment to the project are leveraged.

Self-Organization and Self-Determination

Agile methodologies give development teams the freedom and responsibility to perform many tasks in a self-determined manner. Working habits of developers are strongly connected to their corporate and cultural background. For developers originating from companies or cultures with strict command and control structures, the autonomous work imposed to agile developers, can be a heavy cultural shock and require some time and management effort until adopted. When people have realized the advantages and personal opportunities this autonomy brings, it results in strong motivation and growing commitment to work (Fowler, 2004, pp.4-5). Employees' identification with their job is enhanced, reducing costly staff fluctuation.

5 Conclusion and Outlook

Initially, XP was developed for collocated SD teams. The high demands of XP towards communication among developers as well as between developers and their customers are identified as the major challenges of distribution. The importance of personal relationships as a basis for efficient and effective communication is emphasized. Different possible constellations of distribution are identified and corresponding tools and measures are introduced. New roles are developed to enhance the flow of information and exchange of knowledge between distant development locations. It is argued, in which way the application of tools and measures can compensate the lacking physical proximity and where their restrictions lie. It is also explained how parts of the XP methodology are of particular value to challenges arising from distribution.

In large parts, the different practices of XP can be applied to distributed scenarios. To achieve the communication intensity typical to XP projects, extra efforts have to be brought up. The total abandonment of face-to-face meetings for XP cannot be seen as feasible. If an efficient transmission of information can be achieved, the advantages of agile methodology can be reaped. If the rapid development of increments is severely delayed because of communication lags, the iterative development process becomes inefficient and the core strengths of agile methodology are destroyed. Time differences can be a serious obstacle for agile methodology, because they require the team members to utilize more documentation and tools than XP initially intends. The risk of information lags is amplified. To achieve the vital flow of information extra discipline in the hand-over of information is required. It is up to further empirical research how efficiently XP can be adopted to specific development projects with specific characteristics. In general XP is agile in itself and therefore very adaptable. Even if the approach cannot be applied in total, large elements of it can always be adopted.

XP is a rather young approach and experiences in its adaptation to distributed projects are scarce. The translation, the wide interdisciplinary cognitions delivered by CSCW research, can be of great value to the adoption of agile methodology for distributed development projects. An analysis of the reciprocal influences of agile methodology and corporate cultures can bring valuable insights in the contribution of agile methodology to overall management issues.

List of Literature

[Abrahamsson et al. 2002] ABRAHAMSSON, P.; SALO, O.; RONKAINEN, J.: *Agile software development methods – Review and analysis*. In: *VTT Publications: 478, 2002.*

[Beck 1999] BECK, K.: *Embracing Change with Extreme Programming*. In: *IEEE Computer,* 1999, Vol.32, pp. 70 – 77.

[Beck 1999b] BECK K.: *Extreme Programming explained*. Addison-Wesley 1999.

[Beck et al. 2001] BECK, K.; BEEDLE, M.; VAN BENNEKUM, A.; COCKBURN, A.; CUNNINGHAM, W.; FOWLER, M.; GERNNING, J.; HIGHSMITH, J.; HUNT, A.; JEFFRIES, R.; KERN J.; MARICK, B.; MARTIN, R.C.; MELLOR, S.; SCHWABER, K.; SUTHERLAND, J.; THOMAS, D.: *Manifesto for Agile Software Development.* At: `www.agilemanifesto.org.`

[Boehm 2002] BOEHM B.: *Get Ready for Agile Methods.* with Care. In: *IEEE Computer*, 2002 Vol. 35, pp. 64-69.

[Boehm and Turner 2004] BOEHM, B.W.; TURNER, R.: *Balancing Agility and Discipline – A guide for the Perplexed*. Addison Wesley, 2004.

[Braun et al. 2001] BRAUN, A.; BRÜGGE, B.; DUTOIT, A.: *Supporting Informal Meetings in Requirements Engineering.* In: *7th Internaltional Workshop on Requirements Engineering for Software Quality.*

[Cao et al. 2004] CAO, L.; MOHAN, K.; BALASUBRAMANIAM RAMESH, P.X.: *How extreme does Extreme Programming Have to be? Adapting XP to Large-scale Projects.* In: *Proceedings of the 37th Annual Hawaiian International Conference on System Sciences 2004.*

[de Souza et al. 2002] DE SOUZA, C.R.B.; PENIX, J.; SIERHUIS, M.; REDMILES, D.: *Analysis of Work Practices of a Collaborative Software Development Team*. In: *International Symposium on Empirical Software Engineering 2002*, pp. 3-4.

[Fowler 2004] FOWLER, M.: *Using an Agile Software Process with Offshore Development*. Working Paper 2004.

[Grudin 1994] GRUDIN, J.: *Computer Supported Cooperative Work: Its History and Participation*. In: *IEEE Computer* Vol. 27, pp. 19-26.

[Highsmith and Cockburn 2001] HIGHSMITH, J.; COCKBURN, A.: *Agile Software Development: The Business of Innovation*. In: *IEEE Computer* (p. 120 – 122), 2001, Vol.34.

[Hofstede 1994] HOFSTEDE, G.: *Cultures and Organizations.* HarperCollins 1994.

[Hofstede 2001] HOFSTEDE, G.: *Lokales Denken, globales Handeln – interkulturelle Zusammenarbeit und globales Management*. München 2001.

[Karlström 2002] KARLSTRÖM, D.: *Introducing Extreme Programming – An Experience Report*. In: *XP 2002 – Third International Conference on eXtreme Programming and Agile Processes in Software Engineering,* Sardinia, Italy 2002.

[Kontio et al. 2004] KONTIO, J.; HÖGLUND, M.; RYDÉN, J.; ABRAHAMSSON, P.: *Managing Commitments and Risk: Challenges in Distributed Agile Development*. In: *Proceedings of the 26th International Conference on Software Engineering 2004.*

[Martin 2000] MARTIN R.C.: *eXtreme Programming - Development through Dialog.* In: *IEEE Software*, 2000, pp. 12-13.

[Martin and Schwaber 2004] MARTIN, R.C.; SCHWABER K.: *The Primavera Story. An Agile Transition*. Whitepaper. At: `www.controlchaos.com`.

[Rumpe and Scholz 2002] RUMPE, B.; SCHOLZ, P.: *A manager's view on large scale XP projects*. In: *Third International Conference on Extreme Programming and Flexible Processes in Software Engineering 2002,* pp. 158-159.

[Schlichter 2002] SCHLICHTER J.: *Computergestützte Gruppenarbeit*. Lecture Script, TU Munich August 2002.

[Simons 2002] SIMONS, M.: *Internationally Agile*. At: `www.informit.com`, Person Education, Inc. InformIT.

[The Standish Group 1994] Authors unknown: *The Chaos Report* (1994). The Standish Group International, Inc. At: `www.standishgroup.com.`

[Wallace et al. 2002] WALLACE, N.; BAILEY, P.; ASHWORTH, N.: *Managing XP with Multiple or Remote Customers* In: *XP 2002 – Third International Conference on eXtreme Programming and Agile Processes in Software Engineering,* Sardinia, Italy 2002.