Agile Methodologies for Distributed Collaborative Development of Enterprise Applications

Tobias Hildenbrand and Michael Geisser and Thomas Kude and Denis Bruch and Thomas Acker
Department of Information Systems I
University of Mannheim,
68131 Mannheim, Germany
{hildenbrand, geisser, kude}@uni-mannheim.de, {dbruch, tacker}@rumms.uni-mannheim.de

Abstract—Managing large distributed software projects for enterprise applications with traditional methodologies designed for collocated teams often leads to high planning and management overheads. The sequential and plan-driven traditional approaches often do not allow for an adequate reaction to changes in requirements. Today, Extreme Programming (XP) is the most popular agile development methodology. This paper analyzes how and to what extent XP can be transferred to distributed development projects for large enterprise applications. The focus is on XP in particular, since it is the most common agile methodology in practice and has the highest congruence to the original Agile Manifesto.


I. INTRODUCTION

TRADITIONAL processes are effective especially for the application on critical systems due to its structured development process [1]. Strong and continuous documentation are of significant importance to avoid misunderstandings between the stakeholders involved. However, missing flexibility and long time-to-iterations have led to the development of methodologies that support rapid software development. Especially in medium-sized companies, a flexible methodology supporting changes of system requirements during the development process should substitute heavyweight approaches.

XP, today’s most popular agile development technology, focuses on communication, simplicity, and feedback to improve the speed and quality of software development (SD) [2]. Close communication and cooperation between developers and the continuous integration of stakeholders are key components, neglecting formal processes and documentation to the favor of tacit knowledge to improve flexibility.

According to Sommerville, agile software development (ASD) faces difficulties in its execution, especially in a distributed environment: Customer involvement in the development process becomes a difficult task. Intense involvement between the developers is also typical for agile methods, but may suffer from cultural differences within the team. Moreover, contracts between outside companies are usually based on the software requirements document [1]. Agile methods specify the software requirements document incrementally, leading to difficulties for this contract type. Even though the agile methodology is widely recognized as being especially suited for small- to medium-sized companies, its benefits are convincing and should not be ignored by global companies developing large-scale systems. Therefore, this paper analyzes to what extent XP can be transferred to larger distributed endeavors and its applicability for distributed collaborative development of enterprise applications.

In section II ASD is introduced with a brief description of XP’s principles and practices. Section III systematically analyzes the distributed application development scenario and its characteristics. An overview over a selection of groupware that is supporting project communication is given. In section IV, XP methodology is confronted with the characteristics of distributed application development and the suitability and effectiveness of available tools and techniques is discussed. Section V summarizes the results and gives a short conclusion.

II. 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 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 nine 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 driver’s license database, a new accounting package and an entry system. Amongst the top ranked success factors, respectively reasons for difficulties and failures were the quality and stability of requirements, the quality of plans and estimations, and the degree of user involvement [3].

Agile software development 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 agile philosophy consists of four basic values: Individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan. The first value addresses the importance of direct communication between the involved individuals and the utilization of tacit knowledge as the most efficient and effective ways of error detection, ad-hoc coordination and product improvement [4]. 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. The third value emphasizes the importance of customer collaboration and integration. Software is usually very customer specific, hence project success is strongly dependent on the collaboration of knowledgeable customer representatives [5]. Client-side feedback is meant to drive the evolution of requirements and is essential for the achievement of the overall project mission. This leads to the last value of responding to change. Agile approaches aim at continuous customer satisfaction throughout the whole development life cycle. Changes in requirements, scope or technology are uncontrollable for the development team; hence the only viable strategy is to keep the costs of change as low as possible [4]. Agile methodologies promote lean documentation, incremental and iterative development and simple design, thus keeping the architecture flexible while minimizing the amount of outdated documentation and code.

Today, XP is the agile methodology which is receiving most public attention [6] and which possesses the highest coverage as regards publications. The XP development cycle consists of six phases [7]. In the exploration phase, the development team gets familiar with its development environment and the addressed technology, while the customers write the stories to be implemented which are maintained and expanded throughout the entire development lifecycle [8]. 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 [7] and programmers produce the functionality. Small releases reduce the risk of misled development. XP coding always begins with the development of unit tests. After the tests are written the code is developed, being continuously integrated and tested. At the end of the iteration all functional tests should be running before the team can continue with the next iteration [7]. When all iterations scheduled for a release are completed the system is ready for production. The productionizing phase consists of extra testing of functionality and performance. 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. 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 XP methodology can be understood as development through constant dialog. Developers communicate among 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 customer’s trust and confidence [9].

III. DISTRIBUTED SOFTWARE DEVELOPMENT

SD demands a high degree of collaboration between the involved experts. Software developers spend large parts of their time working with others [10]. As size and complexity of development teams rise, the need for effective communication and coordination grows [11]. When SD projects are distributed over different buildings, countries or continents, communication is aggravated by spatial, temporal and cultural factors. In the following, different reasons and constellations for the distribution of projects and the implications of distribution are discussed. The final part of this section addresses groupware technology supporting communication of distributed development teams.

A. Distribution of Development Teams

Development can be distributed either in a company-internal or a company-external context, according to the parties involved. Company-external collaboration plays an important role in the context of agile methodology, since extensive cooperation with customers is one of its main philosophies. Another example for company-external collaboration is the scenario where different parties of specialists from different companies work on a common goal. These can be projects of equal cooperations with mutual responsibilities, but also sub-projects given to an outsourcing partner. The collaboration with customers is highly important for SD. XP demands a continuous integration of highly knowledgeable customer representatives, where it is often necessary to have parts of the development team working on-site with the customer to collaborate with the rest of the team [11].

B. Implications of Distribution

Teams working in one office 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 [12], [13]. When problems arise, ad-hoc communication or stand-up meetings can be arranged to find a solution. This paper identifies three different issues arising with the distribution of large enterprise application projects: spatial distance, temporal and cultural differences [14]. The spatial separation leads to the exacerbation of communication and coordination. Modern technologies deliver several ways to globally communicate with integrated video and application sharing functionality. Still, social aspects can
make a large difference between medial and direct communication. Verbal communication is enhanced by body gestures and face expressions which are hard to be assessed in medial meetings. Real life meetings also allow the colleagues to build relationships, which is very hard to achieve working in virtual environments (cp. [15]). Working in different time zones is also a challenge for teams developing software in globally distributed scenarios. Communication is often subject to long response times and are restricted synchronous communication due to the lack of overlapping working hours. The impact of time differences on cooperations between the USA and India, for instance, leave only very few hours where project members are in office on both sides. Asynchronous working hours can lead to special impediments for SD projects, especially when development is expected to proceed rapidly (see also [16]). The issue of cultural differences does not directly result from distribution, but it is most likely to accompany many distributed scenarios. Being the most important mean of communication, the language is the most obvious cultural characteristic. A common language is indispensable for the success of a project collaboration. Customs between different cultures can also easily lead to misunderstandings and complicate communication.

C. Communication and Groupware

Groupware or computer supported cooperative work (CSCW) systems refer to technology supporting groups simultaneously working on a common goal. CSCW systems equip the group members with means of communication and interfaces to a shared working environment. Thus fostering group awareness, i.e. the knowledge about the current activities of other group members [13]. 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 [17]. These goals are remarkably symmetrical to the ones of XP’s agile methodology.

Tools that are facilitating communication are particularly important in this context, as agile methodology is very communication intensive. Basic information exchange occurs primarily through messaging systems. Messaging systems can be subdivided into asynchronous messaging in the form of emails and synchronous messaging, realized through instant messaging (IM) systems. Though being well established for communication, both systems do not allow for exchanging emotions. This can be achieved using audio conferencing and video conferencing systems, while the latter offers the closest simulation of collocated meetings [18]. Other collaboration tools are wikis and screen sharing software, which is often used in combination with audio and video conferences. Continuous integration tools monitor status and changes of developed code. Issue tracking systems manage development issues that cannot be resolved immediately [11].

IV. 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 initial form, demands two developers physically sharing one set of computer hardware. XP’s strong integration of customers demands the continuous on-site availability of a customer representative. This is only feasible if the distance of the customer’s location to the development team is relatively small. Since the distance to the main development site can be very large, the local presence of a customer representative is often not viable. However, 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 strongly by distribution. Other XP development practices concerning distributed scenarios are also examined in the following. In doing so, we distinguish remote project participants in near-site and off-site customers and team members, respectively (cp. also [19]). Near-site indicates that the workplace is not shared, but distance can be traveled with only little effort. Stakeholders being off-site are geographically highly distributed, work in a different time zone and might have a cultural background different from that of other project stakeholders.

A. Team Communication

Software developers spend a very large part of their time collaborating with others [10]. The essential role direct communication takes in XP leads to additional project risks when development teams are physically distributed [20]. Open communication requires trustful relationships. Collocated teams communicate heavily while working together or informally crossing each other in cafeterias, hallways or elevators. They constantly exchange business matters and also share some private issues. Valuable personal relationships as foundations for future 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 suggests mutual seeding and maintaining visits between the distant locations to build and maintain relationships as a basis for efficient communication. 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 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 [21].

For the main part of the development life cycle the majority of developers are located at their home development sites. Similar to the moderators in large XP development projects [6], 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. 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-locational 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 directly facilitate the exchange of informal and tacit knowledge [21].

The introduced measures are important for both off-site and near-site scenarios. However, in near-site development collaboration, issues related to team spirit and team awareness 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 in off-site scenarios. Relationships and insights are improved by exchanges in near-site scenarios. However, as in off-site arrangements, efficient collaboration can be tremendously improved if the possible roots of communication deficiencies are considered early on.

The open workspace required for the XP methodology is not imitated sufficiently by the these 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 an adequate means for close collaboration. If availability information is maintained consistently, developers can approach their distant colleagues 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 solved efficiently through messaging, integrated VoIP telephony function or a regular telephone can be used. 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 make them more personal. Especially for people who are not familiar with one another, video conferencing can leverage the effectiveness of communication processes tremendously. 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 [22]. This could be an effective simulation of collocation, leading to efficient osmosis of information and better personal contacts between distant team members.

If teams are developing in time zones with very little or no intersection of working hours, e.g. USA and India, there are almost no occasions for synchronous communication. Severe time differences have significant 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. Wikis can provide space to publish certain basic rules and conventions to enable 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. However, 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 [22]. 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. 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.

B. 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 hire. In distributed development projects this difficulty is accompanied by additional aspects [14], [23]. Large parts of the development are 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 [19]. 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 phase 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 [19]. 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 [19].

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. For them the customer’s reaction time on feedback requests is essential. Due to the narrow scope of small increments, development progress can be severely delayed if response times are long. Next to speed, the quality of information exchange is highly important to get issues resolved correctly in the first approach. 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 emails 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 is 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 [24]. 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. [19] 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.

C. Other XP Development Practices

Some of XP’s remaining practices are very beneficial to the success in distributed enterprise application development scenarios. In the following these practices are examined:

a) 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 (see also [11]). The continuous integration of small bits only leads to small 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. Additionally automated integration tools can generate reports of recent changes, supporting the handover of work between distant development sites with hardly any possibilities of synchronous communication (cp. [21]).

b) Small Release Cycles: 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. Trust and confidence in the development team, as well as the stakeholders’ commitment to the project are leveraged (cp. [11]).

c) 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 (see [21]). Employees’ identification with their job is enhanced, reducing costly staff fluctuation.

V. Conclusion

Initially, XP was developed for collocated SD teams and relatively small projects. However, the analysis of the applicability and benefits of using ASD for the distributed development of larger enterprise application has shown that agile features could be a viable solution to the problems experienced in these types of projects nowadays. The high demands of XP towards communication among developers as
well as between developers and their customers are identified as the major challenges of distributing ASD activities. The importance of personal relationships as a basis for efficient and effective distributed communication is a major issue identified in this paper. Different possible constellations of distribution are analyzed and corresponding tools and measures are introduced and discussed. 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 to foster the applicability of XP and ASD in general in distributed settings. On the other hand, we also discuss where their restrictions lie and what can be done to further introduce agile methodologies in commercial enterprise application projects. It is also explained how parts of the XP methodology are of particular value to challenges arising from distribution.

As a conclusion of our analysis, most of the XP practices can be applied to distributed scenarios. However, to achieve the intense communication typical for XP projects, extra efforts have to be made. An application of XP principles without any face-to-face meetings is not 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 an agile methodology are undermined. 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. Moreover, the risk of information lags is amplified. To achieve the vital flow of information extra discipline in the hand-over and central management 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.

Therefore, the application and, beforehand, adequate adaptation of agile practices can be regarded as a viable approach to alleviating the issues commonly seen when developing enterprise applications collaboratively and spread over various sites. Both the lag of direct interaction necessary for XP as well as the proper documentation and abidance to plans as required in commercial projects can be supported by existing software engineering and collaboration tools.

REFERENCES