CHALMERS

Positive and Negative Quality Effects in Distributed Scrum Projects

An industrial case study

Master of Science Thesis in the Programme Software Engineering and Technology

ERIK TEVELL
MATHIAS ÅHSBERG

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Göteborg, Sweden, June 2011

Positive and Negative quality effects in Distributed Scrum Projects
An industrial case study

ERIK TEVELL
MATHIAS ÅHSBERG

Examiner: Associate Professor Helena Holmström Olsson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2011

Positive and Negative Quality Effects in Distributed Scrum Projects: -An Industrial Case Study

Erik Tevell, Mathias Åhsberg
Chalmers University of Technology
Software Engineering and Technology
{tevelle, ahsberg}@student.chalmers.se

Abstract

As more and more software development projects utilize agile methods in a distributed environment, new process models have emerged. Distributed Scrum has increasingly gained popularity and best-practices state that the Integrated Scrums model in combination with eXtreme programming practices should be applied to maximize the software quality. However, it remains unclear if it is suitable for all types of distributed projects.

Previous case studies within the field have focused on how to implement Scrum in a distributed environment and code velocity compared to other process models. However, rather few studies have taken the quality effect as a viewpoint.

This paper presents a combination of a literature study and an industrial case study with the purpose of finding aspects and practices that effects the software quality in a distributed project where Scrum is utilized. This was done through a root cause analysis with an Ishikawa diagram as a tool, where the quality effects were mapped to different activities. Aspects that have had positive or negative quality effect in the project examined were analyzed and compared with literature. In addition, challenging aspects in a distributed environment are discussed and recommendations are presented.

Finally, this paper concludes that when there exist a substantial amount of non-standardized legacy code, it is preferred to utilize the Scrum of Scrums model due to the decreased dependency between the Scrum-teams. This, to fully benefit from the positive quality effects that Scrum provides.

1. Introduction

Scrum has for the past decade gained popularity as an alternative to organize the development of software [28]. In combination with benefits of offshoring or outsourcing, this has lead to distribution of otherwise co-located teams [15]. Scrum is originally designed to reach hyperproductivity for teams not geographically spread and increase the software quality. However, distributed projects have increasingly implemented this process with various results [29]. Distributed Scrum is a relatively new phenomenon and previous studies have mainly focused on how to implement Scrum methods in a distributed environment. However, there exist rather few published studies that examine the software quality effects of using Distributed Scrum.

This paper presents an industrial case study which analyzes the software quality effects when introducing Distributed Scrum in a product organization. In addition, the software quality attributes correctness and understandability are especially considered. Moreover, best-practices within the field state that the Integrated Scrums model combined with eXtreme Programming (XP) practices can reach the same productivity and software quality as co-located teams [27].

The research questions for this paper are if the Integrated Scrums model is applicable in a distributed project with a legacy code constraint and how agile practices can affect the software quality in a distributed environment.

The disposition of this study is structured as follows. First, deeper knowledge within the distributed Scrum field, agile practices, quality attributes, and related work are presented. Later the case study method is described followed by a presentation of the main findings. Finally, the results are analyzed and positive and negative quality effects when introducing Distributed Scrum are presented.

2. Background

The following section first presents definitions of software quality relevant for this paper followed by a background to distributed software development. In addition, agile practices and their quality effects are described.

2.1. Quality

Defining quality is a rather complex task due to the fact that there exist a wide range of explanations of the term. The International Standards Organization (ISO) defines quality in their ISO 9000 standard as *"the totality of characteristics of an entity that bear on its ability to satisfy stated or implied needs"*. Stated needs are referred to as requirements set by the customer, whilst implied needs are requirements identified by the contractor. Moreover, Juran [12] defines quality as those features that meet customer needs and argues that quality is both oriented to income and costs. Higher quality will require further investments and has therefore an increased cost. However, high quality also leads to less rework and lower customer dissatisfaction, which will contribute to a lower cost.

Even though the definitions of quality stated above can be applied within the software engineering field, more specific clarifications of software quality exist. Meyer [17] divides quality into different attributes, e.g. correctness, reliability, usability etc, which together represent the overall quality of the software. In addition, Sommerville [25] claims that software quality is a management process to minimize defects and ensure that the software reaches the required standards for the quality attributes.

When categorizing different software quality factors, they are divided into two main groups; internal- and external quality [2, 9]. The internal quality focuses on the code itself, e.g. portability or maintainability, and are described from the developers‘ point of view. External quality attributes on the other hand, focus on quality from the users‘ point of view, e.g. correctness and reliability.

The *correctness* quality attribute is defined by McCall et al. [16] as the *"extent to which a program satisfies its specifications and fulfills the users mission objectives"*. Priestley [22] argues that correctness shall be viewed both as the relationship between the program and its specification and as the relationship between the program and its users. The process of checking that a software corresponds to its specification is commonly known as *verification*, while *validation* ensures that software artifacts meet the actual requirements from the users [31].

Finally, McCall et al. [16] state that the *understandability* attribute is defined by how well both the overall logic of the software and the code itself are understandable. They further argue that when using, maintaining or changing the software good insight is required.

2.2. Outsourcing and Offshoring

Even though agile development processes are not designed for distribution, they have been implemented in several outsourced and offshored projects [27]. The definition states that instead of using own resources, *outsourcing* means that an external company performs these tasks or services. *Offshoring* on the other hand, means that resources located in another country are used, but still belong to the same organization. Furthermore, outsourcing and offshoring are often associated with India or China [15].

There are several reasons why companies choose to outsource or offshore part of their development. However, some motivations are more common. First, to reduce costs through cheaper labour. The average hourly rate in developing countries, e.g. India or China, can be less than 10 percent of those in western countries and IT-projects can have decreased their labour cost by 40 percent after three years [15]. Second, to be able to capture talent not available locally and access specialized skills or facilities. Such resources could be more costly locally and may only be needed temporary or occasionally. Finally, the projects can more easily increase the development speed and reduce the time to market [26].

Close collaboration and instant communication are the basic philosophies of agile software development methods. However, these aspects might be impaired by introducing offshoring or outsourcing in agile projects [27].

2.3. Distributed Scrum Models

Scrum is a software development process that also takes project management into consideration. This, through both managing and coordination of resources, and through short- and long term planning [15]. Scrum is especially designed to increase both speed of project progress and flexibility in product development, whilst also improve the overall quality. In combination with other agile development practices, the software quality could be improved even further [3]. Short iterations and direct communication are concepts that Scrum empathizes. This will minimize the risk of delivering a product that does not meet changing requirements introduced during development, hence improve the correctness. Furthermore, these concepts will contribute to faster reaction in a rapid changing environment [7]. In addition, Scrum is designed to enhance energy, focus, clarity and transparency to software development teams [30], which makes it well suited for distributed projects.

The combination of outsourced or offshored development and Scrum is often referred to as *Distributed Scrum*, which can be implemented in three different ways [28].

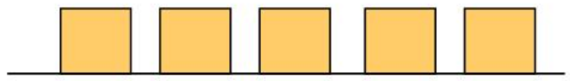

Figure 1. Isolated Scrum-teams [30]

First, *Isolated Scrums*, where the teams are isolated from each other geographically. Often the offshored teams are neither cross-functional, as illustrated in Figure 1, nor uses agile development processes. The requirements can for example, be written in one isolated team, sent to another for implementation, and tested by a third team. This has been proven to cause significant communication problems between the sites [28].

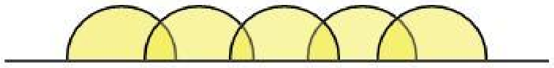

Figure 2. Scrum-teams integrated though a Distributed Scrum of Scrums [30]

Second, *Distributed Scrum of Scrums* [30], where the teams are distributed geographically, but unlike the Isolated Scrums, they are synchronized through overhead Scrum of Scrums meeting with all the ScrumMasters from the different teams of the project. The project‘s tasks are divided between the cross-functional but geographically isolated teams, linked together and coordinated through the Scrum of Scrums. This is the recommended practice by the Scrum Alliance [28]. As Figure 2 shows, the teams are geographically isolated but integrated by the Scrum of Scrums meeting, represented by the overlapping areas.

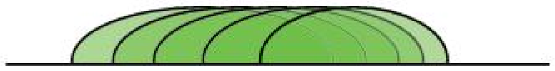

Figure 3. Totally integrated Scrum-teams [30]

Finally, *Integrated Scrums* [30], where all teams are both geographically distributed and cross-functional, as visualized in Figure 3. Each Scrum-team have members at multiple sites, i.e. each employee is both part of a distributed team and a physical workplace. This can lead to increased transparency and information distribution with performance close or equal to co-located teams. The integrated Scrum model is recommended to be implemented by experienced Scrum-teams at multiple sites to function accordingly [30].

2.4. Scrum practices

Scrum also takes some project management aspects into consideration, where three practices focus on planning, implementation and follow-up. First, the *daily Scrum* or the daily stand-up, were each Scrum-team starts their day with a 15 minutes meeting containing project status update [27]. All members answer three questions; “What have you done since yesterday? “, “What are you planning to do today? “ and “Do you have any problems that would prevent you from accomplishing your goal? “. This practice have shown to increase the transparency and enhance the prioritizing of work, which can lead to an increased understandability of the software [8].

Second, in the beginning of each Sprint cycle the *Sprint planning meeting* is held[27]. The team selects the tasks that should be implemented during the next Sprint and completes the Sprint backlog with details. Moreover, the Sprint planning meeting should not take longer than eight hours. This practice enhances the overhead of the Sprint and increases the correctness of the software [21].

Finally, at the end of each Sprint a four hours long *Sprint review meeting* is conducted [27]. Here, the Scrum-team reviews the work that was or was not completed and presents a demonstration to the stakeholders of all the implemented tasks. In addition, this practice adds continuous feedback from the customers after each Sprint, which enables faster adaptation to their current needs, i.e. enhancement of the software‘s correctness [21].

2.5. XP-practices

There are several agile practices used in the combination of Scrum and XP. In the following section the most common ones [14] are briefly described together with their software quality effects.

First, *small releases* is one of the most fundamental agile practice where the development cycle is reduced as much as possible [3]. At each release only the most valuable business requirements are implemented. In addition, the release should be a complete and working software where no included features are only partly implemented. Furthermore, since customers have an ability to make changes during the development to match their current needs, it can be challenging to deliver a product that is correct. However, the short cycles enable a possibility to respond to modifications in the requirements, which leads to an enhanced correctness of the product [18].

Second, the practice of *customer involvement* emphasizes that the customer should be part of the team and work close to the developers. This, to gain both faster feedback during the development and enable customers to continuously perform acceptance tests [3]. Customer involvement will lead to an improved correctness of the deliverables [18] and speeds up the development with faster response to unclear features [3].

Third, the *collective code ownership* practice, where anyone who sees an opportunity to improve or add value to any portion of the code, is required to do so at any time. It is in everyone‘s best interest to continuously improve the software, which leads to a collective responsibility and an enhanced knowledge sharing. This, generating better internal software quality [3].

Forth, implementing the practice of *coding standards* implicates both following defined coding rules and to use standardized techniques. This improves both the understandability of the code within the project and enhances the possibility to practice collective code ownership [3].

Fifth, the *test-driven development* practice roughly states that developers first write the unit-test for a feature, then only implement the code necessary to pass that test. This, leading to both an improved simplicity of the code and an enhanced testability of the software [3]. Several case studies evince that the external software quality can be substantially improved by implementation of this practice. Results have shown that the number of defects reported by customers have decreased by up to 90 percent [5, 19]. Furthermore, Sfetsos et al. [24] conclude that a majority of the published studies related to the subject have stated that the practice will mainly improve the external quality.

Sixth, *continuous integration*, where small pieces of code or features are frequently integrated and tested against the existing software. This practice inverse the traditional way of applying quality control after completing the development, by continuously perform test activities [3]. Moreover, this will detect bugs at an early state by automatically running the test-suit on a regular basis. Continuous integration will decrease time spent on bug-tracking and make it possible to resolve critical issues before the software is released, which in turn will lead to an improved external quality [10]. However, the success of using continuous integration is heavily dependent on how thoroughly the software is tested.

Finally, the practice of *pair programming* states that two programmers share one computer and work together on one task at a time, i.e. one person writes the code and focuses on the implementation of the feature, whilst the other person inspects the code and thinks more strategically and overhead [3]. Moreover, studies have shown that this practice can improve the internal code quality and decrease the number of reported bugs due to constant code review [4, 6]. In addition, pair programming can especially enhance the quality, in comparison with solo programming, when solving challenging or complex tasks [1].

3. Related work

The following section presents empirical findings relevant for this paper. However, rather few case studies have combined Distributed Scrum with XP-practices and focused on the quality effects. Therefore, the findings are further divided into two separate subsections.

3.1. Distributed Scrum

There have been several case studies conducted within the Distributed Scrum field, where prior studies mainly have focused on how the distributed teams can reach a hyperproductive state [15, 20, 28]. The quality aspects in previous studies are often relatively vague on which quality attribute that is considered. Paasivaara et al. [20] have focused on working software after each Sprint as a measurement of software quality when they describe the perceived quality improvements in Distributed Scrum projects.

Moreover, Sutherland et al. [26, 28, 29, 30] discuss software quality where the focus lies within code-quality, errors per lines of code or number of issues mapped to a specific code-error. Their conclusion, concerning software quality, describes the improvement by using Distributed Scrum contra other development processes. In addition, their main finding is the importance of well working Scrum meetings. They advocate daily stand-up meetings for the distributed Scrum-team, during overlapping work hours, were all members can participate. Furthermore, meetings at the start of the day for the different sites are recommended. Sutherland et al. also claim that this will lead to an increased transparency within the project, balance and understanding of cultural differences, clearer communication, increased team-feeling, and the ability to cope with changing requirements. Tools that have been used to reach a sufficient communication level within the teams are e-mail, IP telephony, and video conferencing. To increase the communication even further and reduce the *us vs. them-feeling*, they recommend to initially let the Scrum-teams be co-located and have continuous visits between the sites throughout the whole project.

In addition, Sutherland et al. [26, 28, 29, 30] emphasize the importance of a well working test strategy. The studies show that best-practices in Distributed Scrum are to let developers write unit tests using test-driven development. A separate test team develops scripts for automated testing and performs, in collaboration with the product owners, the manual testing. Moreover, configuration management with hourly automated builds from one central repository and aggressive refactoring when needed, is recommend.

However, these case studies do not specify how the quality attributes have been measured or, in some cases, even which attributes that have been considered.

3.2. Distributed XP

The Scrum Alliance states, as mentioned i section 2.3, that best-practices within the field are the Scrum of Scrums model [30]. However, recent studies claim that the Integrated Scrums model combined with XP-practices will reach productivity and software quality close or equal to co-located Scrum projects [27]. In addition, some of the practices need adaptation to work properly in distributed projects [29].

Previous case studies within the field have identified some common challenges when adopting XP-practices in a distributed environment. Kircher et al. [13] state that small releases, collective code ownership, coding standards and test-driven development can be performed independent of the fact that the teams are distributed or not. Furthermore, they argue that customer involvement, continuous integration and pair programming need adaptation. Since pair programming originally is designed to be used by co-located developers, Kircher et al. suggests that remote pair programming is practiced instead. This is done through video conferencing and application sharing support in the IDE. In addition, Sutherland et al. [28] point out that cultural differences can introduce collaboration problems with conflicting behaviours, processes, and technologies. Moreover, according to Kircher et al. [13] customers should be integrated with the distributed teams through video link, where the "on-site customers" becomes "virtual on-site customers". Nevertheless, Kussmaul et al. [15] state that since requirements change rapidly in the Scrum process, system- or domain knowledge on the different sites may vary, which is further aggravated by the off-site customers. Finally, continuous integration should be implemented as a central repository accessible by all teams and team-members.

However, these case studies do not focus on the quality aspects of distributing the XP-practices.

4. Methodology

The research presented in this paper is a combination of a literature study and a single case study. Initially a literature review within the Distributed Scrum and quality field were conducted. An industrial case study which mainly focused on positive and negative software quality effects were then performed. This to later make a root cause analysis of quality influencing aspects.

As a result of the literature review an Ishikawa diagram was constructed based on the 6M model [11] generating six main categories; people (man-power), tools (machines), process (method), equipment (materials), environment (mother nature) and measurement. In addition, common challenges found in the literature were placed as possible causes under the different categories in the diagram, shown in Figure 4.

The case study was conducted in a large-scale project through a series of qualitative interviews and document inspections. The project is distributed between Sweden and India and uses Scrum as software development method. The combination of Scrum and offshore resources made the project suitable for the needs of our master thesis, since the introduction of both Scrum and the distributed environment was made simultaneously. The company involved is a large global business and technology company who focuses on both in-house development and consulting, more detailed description in section 4.1.

The data collected in this case study is mainly based on semi-structured interviews, where the questions were originated from the result of the literature study. Ten interviews were conducted with people from separate parts of the project, with various roles and at both sites. This, to gain a broad perspective of the project and to be able to triangulate differences and commonalities. The interviews were mostly carried out with face-to-face meetings, but telephone- and Skype interviews were also performed. Finally, all the interviews were recorded and transcribed. This, to gain deeper knowledge, maximize the data collected and to make the root cause analysis easier to perform. In addition to the interviews, documents for role-descriptions, code checklists, guidelines and protocols from retrospective meetings were also collected and inspected. Furthermore, e-mail conversations have been used to complement missing data and clarify uncertainties.

A data source triangulation [23] of the collected documents and interview results was performed to increase the precision of the research. This, to find commonalities and differences among the various sources. Moreover, the data was analyzed and broken down into different categories corresponding to the Ishikawa diagram from the literature study. The found quality effects were further divided into the root causes to narrow down influencing aspects, which is presented in the result section.

Finally, the positive- and negative effects were analyzed and compared against best-practices within the field. In addition, the Ishikawa diagram was revised with influencing aspects found in the case study, generating a new diagram where the effect of each aspect are presented.

4.1. Case description

The case study was conducted in a large-scale Swedish project at Logica Sverige AB. The company is part of Logica corporate group, which is a global organisation with around 39,000 employees in 36 countries. The Swedish section of the company operates mainly within the business- and technology consulting segments in various industries. In addition to this, they also provide several products that are developed in-house.

The product developed in the project examined in this case study is a market-driven enterprise resource planning software. It was originally developed by Logica as a desktop application in the mid-1980s and since 2000 a web-based version of the software have been offered to customers. The customer base mainly consists of municipal-

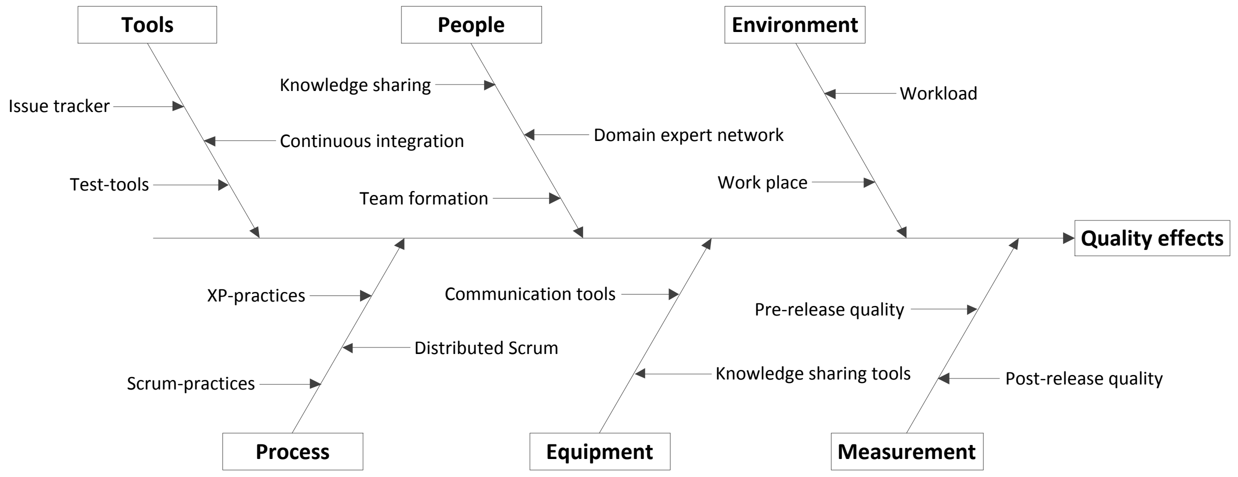

Figure 4. Ishikawa diagram over quality affecting activities based on the literature study

ities and mid- to large-sized companies from a broad range of market segments. The public sector accounts for approximately 60 percent of the customer base, while the private sector accounts for the remaining 40 percent.

The product is divided into several modules that enhance the core functionality. However, all modules are deployed and released as a single product, which then can be customized by the customers to meet their specific needs. Furthermore, new versions of the software are released in a six month cycle, while minor fixes are released on a three week basis.

Historically, an in-house developed programming language have been used for the development of the product. Since the shift towards a web-based platform standard techniques have been utilized. However, a significant part of the product is still dependent on non-standardized legacy code, which complicates the maintenance. In addition, each project includes both further development of the product and maintenance of already released versions and extends over a six month period, which corresponds to the product‘s release cycle.

Prior 2002 different waterfall-methods and RUP have been used with various result within the product development. A decision to implement XP into a pilot project was made in 2002, which was a result of previous search for a suitable agile method for the product organization. According to management, the change of development process in the pilot project lead to an improved software quality and a better overview of the project‘s progress. In addition, developers perceived an enhanced problem solving capability and an increased collaboration. Although the overall experience of the implementation of XP was good, some practices did not suit the team, e.g. pair programming was too intense and some irritation amongst the developers arose. Finally, management experienced XP more as a collection of practices rather than a comprehensive developing process model.

A strategic decision from the group executive board to use offshore resources throughout the whole organization in combination with the previous overall positive experiences with agile developing methods, lead to a demand for a suitable developing process. Distributed Scrum was introduced as an option and initially implemented in a pilot partial project in 2008/2009. An external firm with Scrum expertise was consulted to help with the implementation and customization to fit the product development needs. Furthermore, the process was rolled out incrementally throughout the different teams.

There are approximately a total amount of 85 members included in the project and a majority of them are seen as fixed resources and included in forthcoming projects. The project organization is divided into three major groups; Group A, Group B, and The Automated Test-group, as clarified in Figure 5. Group A and B are further divided into smaller Scrum-teams with 10-12 employees each. Team A1 is an integrated Scrum-team with members in both Bangalore and Gothenburg, whilst Team A2 only is located at one site, the Gothenburg office. In Group B there are three Scrum-teams, Team B1 and B2, located in Sweden, and B3, located in India. Moreover, B2 and B3 develops the same functionality, i.e. share both Product- and Sprint backlog. In addition to these teams, an isolated unit mainly containing offshore resources exists with a responsibility for automation of regression tests. Generally, the Distributed

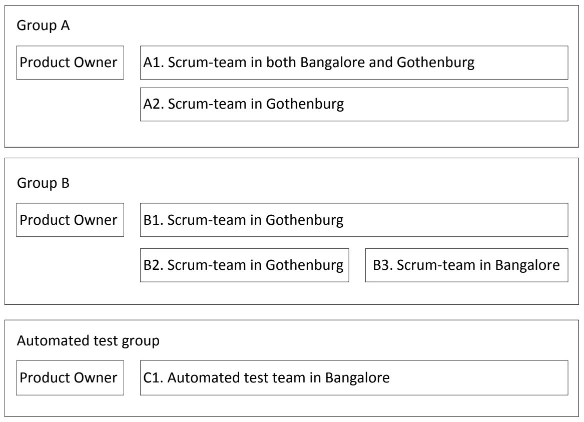
Group A
Product Owner
A1. Scrum-team in both Bangalore and Gothenburg
A2. Scrum-team in Gothenburg
Group B
Product Owner
B1. Scrum-team in Gothenburg
B2. Scrum-team in Gothenburg
B3. Scrum-team in Bangalore
Automated test group
Product Owner
C1. Automated test team in Bangalore

Figure 5. Project organization

Scrum of Scrums model is utilized within the project. However, the Integrated Scrums model is additionally practiced by team A1.

5. Result

This section presents the most significant results regarding software quality from the interviews and inspection of the project documents. The results are separated by the six categories of the Ishikawa diagram from the literature study, as described in section 4.

5.1. Process

Since the introduction of Distributed Scrum there have been several effects on the software quality due to the process change. First, management have perceived an improved correctness of the software by enable cross-functional teams in the product development, e.g. developers and customer support are working together.

Second, management have experienced an increased awareness of both the project- and team progress amongst all levels within the project, which could increase the understandability of the product. This, as a result of the iterative work process that Distributed Scrum provides.

Finally, Scrum of Scrums meetings have been held in each Sprint where both ScrumMasters and product owners have participated. However, the ScrumMaster at the Bangalore site have not attended, which could affect the understandability among members at the Bangalore site. Even though the purpose of these meetings are to transfer knowledge between the teams, they are perceived by the participants as non-productive and are now held more occasionally.

5.1.1. Scrum practices

The practice of *daily Scrum* is emphasised within all teams of the project. Support and management have experienced that the teams‘ awareness of the product‘s current state have increased, which in turn have improved the understandability of the software. Furthermore, management have experienced that the daily Scrum has contributed to an improved issue management and that problems can be resolved when they occur without specific control activities. Moreover, the co-located Scrum-teams hold their meeting in the beginning of the day. The distributed team on the other hand have two meetings per day, where the Bangalore part of the team join the co-located Scrum-team on site at the beginning of their day. In addition they also participate in a daily Scrum with their team through video conference when work hours overlap.

Sprint planning meeting is conducted by all teams within the project at the start of each Scrum cycle. Management, support personnel and developers have perceived that these meetings add an iterative validation that the most important features at the time are developed, i.e. the Sprint reflects customers current needs, which have lead to an improved correctness. Historically, attempts to involve all team-members from both sites through video conference have been made. However, the outcome was not satisfying, which according to management depended on difficulties with containing focus during the whole meeting. Today, team-members from the Bangalore site do not participate during these meetings, instead they receive delegated work from the Gothenburg site.

At the end of each Sprint the *Sprint review meeting* is conducted by all teams of the project. According to management, this practice will increase the awareness of other teams‘ work and the transparency of the project, which could lead to an improved understandability. However, the interest in other teams‘ accomplished work are relatively low, which is indicated by a low participation in the partial project demonstrations.

5.1.2. XP-practices

There are several XP-practices utilized in the project. The following section presents both positive and negative quality effects concerning the different practices.

The agile practice of *small releases*, Sprints in Scrum, is perceived by members at all levels in the project to have increased the overall software quality the most, particularly the correctness. The Sprints are three weeks long and have lead to a series of quality effects. First, management have seen a positive effect with better overview of the project and an increased prediction of the outcome, where both time- and resource management have been experienced as improved. This have lead to a faster response to changing requirements, which in turn can improve the correctness. Second, the iterative testing in the Sprints have resulted in a shift from a massive test-period at the end of each project, towards a validation- and verification phase before the release. It has been experienced by all levels of the product organization that the software is of higher quality when entering the final test period. Finally, according to management the ambition to have executable code at the end of each Sprint have clarified overall project goals, increased the opportunity of producing automated tests and enabled pilot customer involvement at an early state. However, members at all levels agree to the fact that the product is not yet releasable after each Sprint.

According to management and support personnel, the product development has shifted from a technology-driven perspective towards a market-driven focus as a result of an increased *customer involvement*. This have lead to an improved correctness of the deliverables and velocity of the development with faster response to unclear features. End-users have been involved during the development of some customer-specific features. This to both gain feedback on existing functionality and to get suggestions for improvements. In other cases, support personnel have acted as customer representatives due to their domain knowledge. In addition, a reference group with stakeholders have been consulted on a regular basis during the projects to further validate the features of the software.

Collective code ownership is used throughout the whole project. Management and developers have experienced an impaired understandability of the product since the introduction of Distributed Scrum. The majority of the developers, especially at the Bangalore site, do not have sufficient knowledge of the system due to the extensive legacy code base. This have prevented the practice to be used at its full capacity. Moreover, some problems with the collective code ownership at the site in Bangalore are experienced. Affrightment of failure or exposure is mentioned as root causes to this problem. Even though this practice is generally utilized, management argue that it is somewhat dependent on the developers knowledge of the system and domain. However, developers at the Swedish site have perceived that this practice continuously have improved the software and united the responsibility for the code quality.

The practice of *coding standards* is applied in the project. It is experienced by both management and developers that the understandability have been improved since the introduction of Distributed Scrum. The standardization of techniques has increased the opportunity for the offshore resources to learn and gain insight into the system. In addition, documents and checklists are used to ensure standardization of the code and Javadoc is additionally applied for detailed description when needed. The product‘s dependency on legacy code, mentioned in section 4.1, are seen as a major issue by members at all levels within the project. Mainly due to the use of a non-standard development language, parts which are written in Swedish and the complexity of the system. Attempts have been made to translate parts of the code into English to enable involvement of developers at the site in Bangalore. However, this was too resource-demanding and the outcome was not as satisfying as expected.

The agile practice of *test-driven development* is not fully implemented in the project and no experienced quality effects have therefore been noticed. Management have perceived that there exist disagreement within the project whether or not this practice should be utilized. However, some developers have used test-driven development spontaneously. Even though unit- and integration-tests are written, there exist no stated test-strategy within the project. The reason for this is according to management that Distributed Scrum lacks an uniformed best-practice. It is also perceived that better organization and structure of test methods are needed. In addition, management would like to perform a more risk-based test strategy with focus on testing the right things, rather than strive for a high test-coverage.

Pair programming is not used in any of the Scrum-teams within the project. However, there exists an interest of implementing this practice among some of the developers, who believe that continuous code-reviews would enhance the internal software quality of the product.

5.2. People

The integration between support personnel and development team has according to both management and support lead to an improved work-flow for their issue management and an enhanced knowledge distribution, which could lead to an improved understandability. Support personnel are seen as domain experts within the project and have the responsibility to perform acceptance test activities. Due to prior frictions between support- and developing teams, mainly concerning prioritization issues, a decision to integrate these departments has been made. However, the teams in Bangalore lack on-site domain experts and are therefore dependent on the Swedish teams. Management at the Gothenburg site perceive that there is a risk of losing valuable knowledge within the project due to a high degree of staff turnover in the Indian labour market in general, which in turn could lead to a impaired understandability. In addition, they claim that the lack of domain- and system knowledge at the Bangalore site have lead to an uneven distribution of work.

5.3. Environment

Since the introduction of Distributed Scrum, the shift in working environment at the Swedish site has gone from isolated rooms to an open plan office. Management have perceived that this has increased the communication between team-members and added transparency in the project, which in turn could improve the understandability. Moreover, they argue that this have lead to a faster problem solving and decision making in the development, which could lead to an increased correctness of the software due to a more rapid response to customers' current needs.

The distributed environment has however introduced some challenges. Management have experienced that there exists a slowness in the communication between the sites, which could affect the understandability negatively. Developers at the Gothenburg office have occasionally preferred to resolve problems on their own without involvement of the distributed team. Nevertheless, continuous visits between the sites from team-members at all levels within the project have according to management improved the distribution of work.

Furthermore, the internal software quality have been improved according to members at all levels of the project since the introduction of Distributed Scrum. The workload throughout the project has shifted from an escalation close to the release towards a more balanced pace of work. This as a result of the shorter iterations, which is perceived to reduce the stressful work environment.

5.4. Measurement

The post-release quality of the product is measured in several ways within the project. Management and support have experienced that the time spent on non-chargeable support issues has decreased drastically since the introduction of Distributed Scrum. This further confirms that the correctness of the software have been improved. However, management cannot distinguish if the effect is originated from other quality improvement decisions made during the same time period. They believe that the combination of Distributed Scrum and the introduction of automated regression tests have contributed the most to the product's improved quality. Moreover, the issues reported to the support department are used as an indicator for the product's quality. Both the number of reported issues and the time spent by support personnel on each issue are measured. In addition, management have seen an increased development of new features rather than spending time on maintenance, which is another indicator of an improved correctness.

Moreover, surveys are performed regularly with the purpose to get feedback from stakeholders, which according to management enables proactive actions. This can in turn improve the correctness of the software due to a closer customer involvement. These surveys cover both aspects related to the product itself, e.g. user interface, product features etc, and how the administrators experience the quality of the product's supporting services. In addition, the outcome of the surveys are used as input for the development. However, no direct measurements of how the end-users experience the product are performed.

5.5. Tools

The agile practice of *continuous integration* is emphasized within the project. The immediate feedback that this practice introduces have according to developers increased the opportunity to resolve code related issues more rapidly. This have in turn have lead to an improved external software quality. Moreover, unit- and integration tests are run upon code check-in and the product is built each night based on the latest available code. In combination with the build is the application more thoroughly tested and reports with test-results are generated. When the tests fail, a notification is sent by e-mail to the developer who caused the failure. However, the automated regression tests are not included in the nightly builds since they are schematically initialized manually.

5.6. Equipment

The overall communication within product organization have decreased since the introduction of the distributed environment, according to members at all levels at the Swedish site. This could in turn lead to an impaired understandability of the software among the members. To minimize this problem between the sites different tools have been utilized. Although SMS and e-mail are used, the most popular and effective way of communication is according to members at all levels of the project through chat or video-chat. For group meetings videoconferences have been the preferred way. However, management have experienced that the booking of videoconferencing rooms at both sites simultaneously can be an issue and therefore video-chat is sometimes used instead.

Furthermore, project documentation are mainly stored and managed in an issue tracking software, which can be accessed by all project members. Support personnel and developers have experienced an increased knowledge sharing and an enhanced transparency due to the increased information availability. This, could improve the understandability of the product among the members even further. Additional project documentation which are not stored in the issue tracking software are instead managed through a shared folder structure.

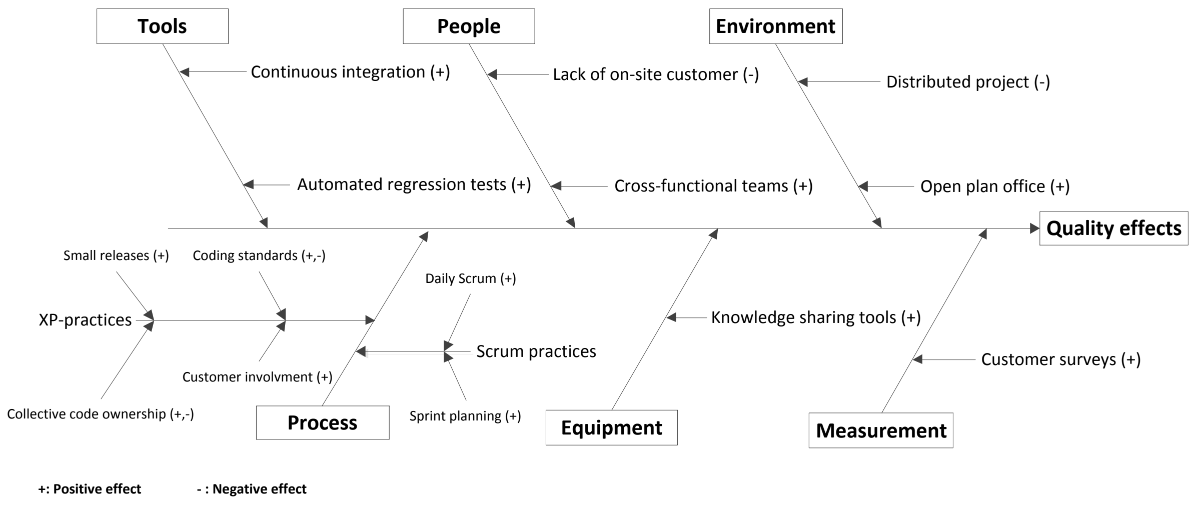

Figure 6. Ishikawa diagram over quality affecting activities based on the case study result

Aspect	E	C	I	U
Continuous integration	+			
Automated regression tests		+		
Lack of on-site customer				-
Cross-functional teams		+		+
Distributed project				-
Open plan office		+		+
Small releases	+	+		+
Coding standards				+
Collective code ownership			+	-
Customer involvement		+		
Daily Scrum				+
Sprint planning meeting		+		
Knowledge-sharing tools				+
Customer surveys		+		

Table 1. Influencing aspects on the overall External quality, Correctness, overall Internal quality and Understandability.

6. Analysis

The following section presents aspects that have had positive or negative effect on the quality of the product examined in this case study, summarized in Figure 6. In addition, challenging aspects that were not fully utilized in the project are presented. Finally, recommendations for how to implement Distributed Scrum in a project with a significant legacy code constraint are stated.

6.1. Positive quality effects

As seen in Table 1, there are several aspects that have had a positive impact on the software quality. First, the overall external quality have been affected mainly by two practices. *Continuous integration* through the immediate feedback to developers, and *small releases* through the introduction of the iterative work-process. The short cycles, employed in these aspects, have enabled continuous test activities throughout the whole development process. Compared to a waterfall process model, this could eliminate the dedicated test period or convert it towards a validation- and verifications phase. Furthermore, both previous case studies and the one examined in this paper show that these practices can be applied in a distributed environment without special adaptation.

Second, Table 1 also reveals that there are several aspects that have positively affected the correctness attribute of the software quality. Since requirements often change throughout a project, literature and previous studies have recommended a close collaboration with customers to minimize the risk of delivering software that does not correspond to current needs. The project examined in this case study performs *customer surveys* to enhance the *customer involvement* and take advantage of the support personnel‘s frequent contact with users. The *cross-functional teams*, where developers and support are united, have lead to a mutual understanding of users‘ needs among the project members. An *open plan office* have further enhanced the collaboration between developers and support personnel by removing communication obstacles. Even though the close customer collaboration is limited to the Gothenburg site, the result shows that the correctness of the deliverables have been improved since the introduction of Distributed Scrum. In addition, *Sprint planning meetings* have enabled an iterative validation of the software and an ability to cope with changing requirements, which have further contributed to an improved correctness. Moreover, previous case studies recommend that automated tests are developed to minimize resource allocation, which can benefit the manual test-coverage. The result from this case study verifies that utilization of *automated regression tests* have improved the external software quality, and in particular the correctness attribute.

Third, the result shows that utilization of the *collective code ownership* practice have improved the overall internal software quality. Previous studies show that this will lead to a united code responsibility, which in turn have a positive effect on the code quality. Even though members at the Bangalore site cannot contribute with code improvements in all parts of the software, due to the legacy code constraint, the overall experience within the project indicates that this practice has had a positive internal quality effect.

Finally, the result further reveals that the use of *coding standards* have improved the understandability within the project, which have enabled new members to enter the project and gain insight of the product relatively fast. Moreover, *Knowledge-sharing tools* and *daily Scrum* meetings are empathized within the project with the purpose to improve the transparency between the sites. Previous studies have shown that an enhanced transparency can lead to an increased understandability, which is verified in the examined case study.

6.2. Negative quality effects

The interviews and document inspections revealed several negative software quality effects of introducing Distributed Scrum, as visualized in Table 1. XP-practices emphasize that a customer or customer representative shall be on-site during the development to ensure the correctness and speed up the development by fast response to unclear features. However, the result shows that it has been difficult to transfer the required system- and domain knowledge to the Bangalore site, mostly due to the system‘s complexity and legacy code dependency. This have made the teams in Bangalore dependent on expertise from the Gothenburg office, which have resulted in an uneven distribution of work. A slowness in communication between the sites have further affected the knowledge-sharing negatively, which in turn have impaired the possibility to increase the overall domain- and system knowledge within the project.

The legacy code dependency has also affected the implementation of the *collective code ownership*. The lack of sufficient system-knowledge at the Bangalore site aggravates these members to apply this practice, which have had a negative effect on the understandability. In addition, cultural differences can affect the utilization of this practice, since not all developers have the courage to make changes in the code. This can in turn also affect the correctness of the software negatively due to less continuous improvements of the code base.

6.3. Challenging aspects

Several recommended practices when using Distributed Scrum have not been utilized at their full capacity within the examined project. First, the result reveals that the practice of *Scrum of Scrums meeting* is not implemented fully in the project and therefore no quality effects related to this practice can be evaluated. This, since the ScrumMaster at the Bangalore site is not involved and the participants at the Swedish site have perceived these meetings as non-productive. The purpose of this practice is to share knowledge between teams to increase the overall understandability. It is therefore recommended that when applying the Distributed Scrum of Scrums model, this essential practice should be fully implemented to maximize the positive quality effects.

Second, *Sprint review meetings* is a Scrum practice that enables inter-team knowledge-sharing through demonstrations, which could add further transparency between teams. This practice is applied in the project examined in this case study, but the result shows that the modest interest in other teams‘ work have lead to low participation in theses meetings. Hence, no quality effect have been experienced. According to previous studies Sprint review meetings could lead to both an improved correctness and an enhanced understandability if applied correctly. It is therefore recommended that this practice should be implemented in a distributed environment to increase the transparency, which in turn could improve the software quality. However, the meetings might need adaptation when overlapping working hours are few or none.

Third, since *test-driven development* is not used in the project, no positive or negative quality effect related to this practice have been perceived. Previous studies have shown that test-driven development could lead to a simplified code, an enhanced testability of the software and significant decrease in number of defects reported by customers. Furthermore, it is shown that this practice is well suited for a distributed environment and it is therefore recommended to fully utilize test-driven development in a Distributed Scrum project.

Finally, *pair programming* is designed to improve both the internal and external software quality. This, through constant code review and an enhanced problem solving on complex tasks. Previous studies have shown that this practice could be implemented in a distributed environment with technical assistance. However, when there are few or none overlapping work hours it is recommended that pair programming is utilized only between co-located team-members. In addition, the result from the case study reveals that friction between team-members can arise when implementing this practice. Based on this fact, it is also recommended that pair programming could be applied more occasionally where closer and more intense collaboration is needed.

6.4. Recommendations

The research question, as stated in section 1, for this paper is whether or not the Integrated Scrums model combined with XP-practices is suitable in all types of projects and organizations. The project examined in this study utilizes a form of Distributed Scrum of Scrums in combination with several XP-practices. However, a change to the Integrated Scrums model is not recommended due to the strong legacy code dependency. Even though it could be argued that this model enhances transparency, the result shows that maintenance of the legacy code base requires language-specific knowledge and that translations are too resource demanding. This prevent members to be involved in all parts of the development, which could lead to an uneven distribution of work and in turn negatively affect the quality. To cope with this constraint it is therefore recommended that members at the offshore site are only involved in development of new features and maintenance of already standardized parts. Hence, the Distributed Scrum of Scrums model are more suitable when there exist a significant legacy code base that are dependent on localized knowledge. The teams are less integrated when this model is used which allows them to work more independently on divided tasks.

7. Validity

The external validity of this study is threaten by the fact that the result is based on a single case study where one product organization has been analyzed under its specific conditions. Furthermore, the project examined in this study have implemented Scrum and introduced a distributed environment simultaneously, which have prevented us to distinguish between the two factors. However, the recommendations given in this study are based on best-practices within the field that have been proven by a number of previous case studies to be successful in a distributed environment. This will minimize the threats to the external validity and support our findings to be applied in other projects with similar conditions.

The threats to the internal validity of this paper have been minimized due to the use of multiple data sources and triangulation. Interviews with all levels at both sites have been performed to ensure that the findings are well established throughout the organization. Moreover, all interviews have been recorded and transcribed to further strengthen the validity of the data collected. Finally, the accuracy of the result has been confirmed by employees within the project.

8. Conclusion

In this paper a combination of a literature study and an industrial case study has been performed to enlighten positive- and negative quality effects when introducing Scrum in a distributed environment.

Best-practices within the field state that the Integrated Scrums model in combination with XP-practices should be utilized for highest software quality. The result from both previous case studies and the one conducted in this paper show that the majority of these agile practices can be utilized in a distributed environment and indeed improve the software quality.

The major findings in this case study reveal that the agile practice of small releases have increased the overall external quality, the correctness of the software and the understandability of the code itself. Furthermore, by expanding the cross-functional teams with support personnel have lead to both an increased correctness and an improved understandability of the software.

However, the result also reveals that some XP-practices could have a modest effect or in some cases even impair the software quality when applied in a distributed environment. A substantial amount of non-standardized legacy code could increase the knowledge gap and prevent team-members to fully utilize all practices. For example, the practice of collective code ownership has not been fully utilized at all sites in the examined project, which have affected the understandability of the system negatively. In addition, the lack of an on-site customer or domain expertise at all sites have impair the understandability of customers needs.

In addition, a non-standardized legacy code constraint could also lead to an uneven workload within a team when an Integrated Scrums model is used. This, as a result of members lacking system- or domain knowledge, which could divide the maintenance of the software and development of new features within the group.

Finally, this paper concludes that the Integrated Scrums model is not suitable in a project with a significant non-standardized legacy code constraint. Instead, the Distributed Scrum of Scrums model is to recommend, which allows teams to work more independently.

9. Acknowledgments

The authors would like to thank the employees at Logica who have participated in this research. A special acknowledgement to Carl Thyberg who made this research possible, and Stefan Svensson for the approval of the thesis and his valuable input. Finally a special thanks to our supervisor, Associate professor Miroslaw Staron, for his advise and support throughout the whole project.

References

[1] E. Arisholm, H. Gallis, T. Dybå, and D. I. Sjöberg. Evaluating pair programming with respect to system complexity and programmer expertise. *IEEE Transactions on Software Engineering*, 33:65–86, 2007.

[2] M. Barbacci, M. H. Klein, T. A. Longstaff, and C. B. Weinstock. Quality attributes. Technical Report CMU/SEI-95-TR-021, ESC-TR-95-021, December 1995.

[3] K. Beck and C. Andres. *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[4] A. Begel and N. Nagappan. Pair programming: what's in it for me? In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, ESEM '08, pages 120–128, New York, NY, USA, 2008. ACM.

[5] T. Bhat and N. Nagappan. Evaluating the efficacy of test-driven development: industrial case studies. In *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, ISESE '06, pages 356–363, New York, NY, USA, 2006. ACM.

[6] G. Canfora, A. Cimitile, F. Garcia, M. Piattini, and C. A. Visaggio. Evaluating performances of pair designing in industry. *Journal of Systems and Software*, 80(8):1317 – 1327, 2007.

[7] H. F. Cervone. Understanding agile project management methods using scrum. *OCLC Systems & Services: International digital library perspectives*, 27:18–22, 2011.

[8] M. Coram and S. Bohner. The impact of agile methods on software project management. *Engineering of Computer-Based Systems, IEEE International Conference on the*, 0:363–370, 2005.

[9] R. Fitzpatrick, P. Smith, and B. O'Shea. Software quality challenges. *IEE Seminar Digests*, 2004(913):6–11, 2004.

[10] M. Fowler and M. Foemmel. Continuous integration. Thoughtworks, [online], 2006. Available from: `http://www.martinfowler.com/articles/continuousIntegration.html(May2011)`.

[11] A. Gwiazda. Quality tools in a process of technical project management. *Journal of Achievements in Materials and Manufacturing Engineering*, 2006.

[12] J. Juran and A. Godfrey. *Juran's Quality Handbook*. McGraw-Hill, 1999.

[13] M. Kircher, P. Jain, A. Corsaro, and D. Levine. Distributed extreme programming. In *Second international conference on eXtreme Programming and Agile Processes in Software Engineering*, pages 66–71, 2001.

[14] H. Kniberg. Scrum and xp from the trenches. Crisp, [online], 2006. Available from: `http://www.metaprog.com/csm/ScrumAndXpFromTheTrenches.pdf(May2011)`.

[15] C. Kussmaul, R. Jack, and B. Sponsler. Outsourcing and offshoring with agility: A case study. *Extreme Programming and Agile Methods - XP/Agile Universe 2004*, 3134:147154, 2004.

[16] J. A. McCall, P. K. Richards, and G. F. Walters. Factors in software quality. volume i. concepts and definitions of software quality. Technical report, Rome Air Development Center, 1977.

[17] B. Meyer. *Object-Oriented Software Construction*. Prentice Hall PTR, 1997.

[18] E. Mnkandla and B. Dwolatzky. Defining agile software quality assurance. *Software Engineering Advances, International Conference on*, 0:36, 2006.

[19] N. Nagappan, E. Maximilien, T. Bhat, and L. Williams. Realizing quality improvement through test driven development: results and experiences of four industrial teams. *Empirical Software Engineering*, 13:289–302, 2008.

[20] M. Paasivaara, S. Durasiewicz, and C. Lassenius. Using scrum in a globally distributed project: A case study. *Software Process Improvement and Practice*, 13:527–544, 2008.

[21] M. Pikkarainen, J. Haikara, O. Salo, P. Abrahamsson, and J. Still. The impact of agile practices on communication in software development. *Empirical Software Engineering*, 13:303–337, 2008. 10.1007/s10664-008-9065-9.

[22] M. Priestley. The logic of correctness in software engineering. 2008.

[23] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14:131–164, 2009. 10.1007/s10664-008-9102-8.

[24] P. Sfetsos and I. Stamelos. Empirical studies on quality in agile practices: A systematic literature review. In *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*, pages 44 – 53, 29 2010-oct. 2 2010.

[25] I. Sommerville. *Software Engineering*. Addison-Wesley, 2004.

[26] J. Sutherland, G. Schoonheim, N. Kumar, V. Pandey, and S. Vishal. Fully distributed scrum. *IEEE Computer Society*, 27, 2009.

[27] J. Sutherland and K. Schwaber. The scrum papers: Nut, bolts, and origins of an agile framework. *Scrum inc.*, 2010.

[28] J. Sutherland, A. Viktorov, and J. Blount. Adaptive engineering of large software projects with distributed/outsourced teams. In *International Conference on Complex Systems*, 2006.

[29] J. Sutherland, A. Viktorov, and J. Blount. Distributed scrum: Agile project management with outsourced development teams. In *Agile2006 International Conference*, 2006.

[30] J. Sutherland, A. Viktorov, J. Blount, and N. Puntikov. Distributed scrum: Agile project management with outsourced development teams. *Hawaii International Conference on System Sciences*, 0:274a, 2007.

[31] D. Wallace and R. Fujii. Software verification and validation: an overview. *Software, IEEE*, 6(3):10 –17, may 1989.