CHALMERS

Improvement of “the new agile process for distributed projects”

Master of Science Thesis in the Programme Software Engineering and Technology

MIR MOHAMMAD SAMSUL AREFIN
DENIS KORZUN

Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, September 2010.

The Author grants to Chalmers University of Technology the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.
The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology store the Work electronically and make it accessible on the Internet.

Improvement of “the new agile process for distributed projects”

MIR MOHAMMAD SAMSUL AREFIN
DENIS KORZUN

© MIR MOHAMMAD SAMSUL AREFIN, September 2010.
© DENIS KORZUN, September 2010.

Examiner: SVEN-ARNE ANDREASSON

Chalmers University of Technology
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. September 2010.

Abstract

Over the last decades more and more software development companies transfer at least a part of their development process to the so-called offshore countries. To increase the productivity of these projects "the new agile process for distributed projects" was invented. The research question of this work is how "the new agile process for distributed projects" can be improved. This research uses a case study methodology to identify waste and gaps in the process. Studies of other processes, conducted mainly by literature review, are used to fulfill the identified gaps. As a result of the research several improvements were proposed and questions for the future research were identified.

Keywords: Agile development, Global software development, Distributed projects, SCRUM, ROPES, RUP, NAPDiP.

Acknowledgement

This Master's thesis report has been written as the final part of the Master's program Software Engineering and Technology at Chalmers University of Technology, Sweden. The subject was chosen in collaboration with an IT service company in Northern Europe, where the thesis also has been performed.

First of all we would like to thanks our supervisors Magnus Klack and Stefan Bryne at the company for their guidance and support during this research work. We are grateful for their valuable supervision, motivating ideas and never-ending optimism.

We would also like to thank our examiner Sven-Arne Andreasson from Department of Computer Science and Engineering at Chalmers University of Technology for his valuable advices and support during the work.

Furthermore, we would like to thank those people who helped and supported us time to time during the work whose names are not mentioned here.

We would like to thank you all!

Table of Contents

List of Figures

List of Tables

1 Introduction

Nowadays more and more software development companies transfer at least a part of their development process to the so-called offshore countries (e.g. India, China, Ukraine, Belarus etc.) It is a very common situation where a team working on the same project, but it is scattered around the world. It gives a lot of different benefits to the companies. First of all this type of development allows to decrease the cost of the development significantly because salaries and taxes usually are lower in these countries. Occasionally these countries give preferential tax laws for the software development companies residing there. As well an offshore development provides an access to the talents of the offshore countries. Sometimes it can be hard to convince the specialist to move to another location or to another country. On the other hand, if the company has a department in the country and the position that they would offer would look more attractive.

However, running this kind of projects is a very challenging affair because multinational and multicultural projects have to deal with lots of issues that a collocated project does not have to deal with. First of all, the huge physical and cultural distances make an enormous impact on Global Software Development (GSD). For example several nationalities can have similar culture, language and traditions, even common history. If foreign customer would call them by the same name, according to their shared history they can be stressed as the questions of the national self-consciousness can be important for them and it can cause negative effect on the future cooperation. Another example is traditions and ideology caused by religion, proposition of a pool party for a mixed gender group as a team building activity can offended the members of a traditionally religious country. Physical distances put significant limitations on the communication process. Even now with availability of phone- and video-conferences the developers can't be relaxed communicating with another site, the developer speaking with his manager may hesitate to ask about unclear statements. Second time, as he would not want to show that his English is not excellent. The greatest challenge in many offshore projects would be to communicate complex problems with the offshore sites [1]. This will sometimes resulted in "hacked" code and violations against the product architecture. Other architectural related matters concerned the difficulties to benefit the architecture quality attributes such as performance, maintenance, usability, scalability and testability [1].

On the other hand during the last years agile development processes have shown good results in industrial projects. There are several agile processes adapted to the distributed development (Scrum for example). Running a global project in an agile manner provides significant benefits like reduction of delivery time, boosting knowledge exchange, steadiness to requirement change, continuous delivery, interaction with customer, fast feedback from teams, sustainable development, test driven development (TDD) benefits. But there are certain drawbacks of a "pure" agile approach to the global software development. Projects running in an agile manner with minimal documentation will eventually run greater risk of frequently breaking the architectural rules. If the architectural rules are not followed, the application tends to turn into a piece of hard working material, inviting further violation to the architecture. Soon the application will be unnecessarily difficult to maintain resulting higher cost, lower quality and longer time spent for each additional fix. In this master thesis we focused on the improvement of "the new agile process for distributed projects (NAPDiP)". This process has taken the most appropriate features and best practices from Rational Unified Process (RUP), Extreme Programming (XP), Scrum and Rapid Object Oriented Process for Embedded System (ROPES) [13], [30]-[35].

NAPDiP is a delivery model and a way of working with agile global outsourcing. It is an iterative process and each of the iterations is divided into five phases and eighteen steps in total. The phases are analysis, design, implementation, test and advance. This process doesn't include initial requirement analysis as part of the development phases. It assumes that the team is already gathered and initial requirements are specified.

The purpose of NAPDiP is to organize the work of international teams while keeping the benefits provided by agile way of development, but at the same time taking into account communicational, cultural and different level of experience issues. As it was already mentioned NAPDiP is a relatively new development process and is therefore eligible for a wide range of improvement.

According to the policies of the Northern European Company where this master thesis was initialized authors are not allowed to mention the "real" name of the company and the "real" name of the process. As well authors are not allowed to mention the names of the projects involved in the case study.

2 Theoretical Background

2.1 Global Sourcing

The term global sourcing is used to describe a practice of sourcing from a global market for the products or services across political geographical borders and boundaries. The main reason for global sourcing is the access to competencies, resource availability and costs benefits. Over the last decade global sourcing has major impact on information technology in terms of software development [1]. The development activities of large enterprises of developed countries are distributed among comparatively low cost locations and organizations [2]. Besides cost-effectiveness; the efficient system performance, minimized risks and time to market in today's integrated comprehensive market are the attributes of success. That's why software development is considered as a globally sourced commodity. It has also facilitated the use of well-educated and technically competent software engineers from low cost countries of Eastern Europe, South Asia and Latin America etc. [3].

Global software development involves interactive people, organizations and technology across the nations with different backgrounds, languages, cultures and working styles. It improves the interactions and relationships between individuals and groups. It helps to solve complex issues in an efficient way. Adaptation of GSD offers organizations very promising benefits [4], [5], [7] and [8]:

- ***Cost Savings:*** The primary benefit of global sourcing is to lower the cost of the development. The price of system development in USA, Canada and most European countries is higher than it is in Eastern Europe or Asian countries.

- ***Access to multi-skilled workforces:*** It opens the possibility to hire skillful and knowledgeable engineers. The companies have the opportunity to expand their software development activities for contribution of large number of skilled workers, wherever they may be located. As well, it enables the creation of a virtual cooperation or teams across the nations.

- ***Reduced time to market:*** It has an ability to use time zone differences to achieve "follow the sun" approach described in details by Carmel [6]. It enables organizations to maximize the productivity by usage of multination resources. It improves the time to market.

- ***Improve quality:*** Sharing best practices among business organizations can improve the quality of the product.

- ***Contiguity to market and customer:*** GSD allows to develop the software in tight proximity to actual customer and to increase the knowledge of the local market. For example an embedded system software development company focused on large manufacturing industry based in China, or Enterprise Resource Planning (ERP) system Development Company focused on the new market in India.

- ***Scalability:*** The outsourcing company usually is prepared for increasing or decreasing the amount of workers either temporarily or permanently.

Beside of these highlighted benefits of the global sourcing, the organizations have other bonuses such as an improvement of resource allocation, reduction of supervision cost, an extension of experience in application of software development and documentation processes.

However, there are some major challenges organizations need to overcome in global software development. Communication, language barriers, social and cultural distances, time zone, trust and coordination problem between teams are important issues of global software development [4], [5] and [7]:

- ***Communication distances;*** it is the significant obstacle in GSD. There are a lot of problems caused by lack of communication among the developers. As well a large amount of development time is spent for communication. Despite of different technologies available for communicating between teams, for example email, Instant Messaging (IM), phone and video conferencing, time differences between two sites create barrier for quick communication.

- ***Language barrier;*** language difference is another major barrier of GSD. Despite of usage of English as a common language, difficulties in understanding between the parts of the teams caused by educational and cultural differences still exists [4].

- ***Social and Cultural differences;*** it makes difficulties in GSD. For example, people in Belarus or India follow and maintain the hierarchy where swedes emphasis on equal representations [4].

- ***Trust and confidence;*** sometimes offsite part of the team does not complete the tasks in a certain period of time or finishes the task in a different way because of misunderstanding of the requirements provided by the onsite part. If it happens periodically, the onsite part has difficulties to establish a feeling of trust with the offsite and the lack of trust between two sites reduces the confidence [4].

- ***Coordination and control;*** this issue arises when the project tasks are divided between the parts of the team. If both sites try to control the project then there will be a conflict between the parts of the team. The project could be delayed because the lack of management between the team parts and sites [4].

2.2 *Agile Development*

Over the last years agile development processes has shown a good results when applied in software development projects. As a result more and more projects are uses these processes.

2.2.1 The Emerging of agile development processes

Agile software development in its current meaning appeared in mid-1990s as an alternative to the standard (of that time) heavily regulated and strictly document based software development processes. That time in the developer's community emerged the theory that traditional processes require huge amount of paperwork, these papers and artifacts slow down the development and are sometimes even useless. Another very important issue for that time was inability of the customer to define the final set of the requirements and features for the software from the start of the project which caused changes during the development process. A traditional heavily regulated and regimented development model (for example the waterfall model) [9] was not able to handle this issue properly. Due to that, the focus in the development process was switched from fulfilling clearly stated project requirements to continuous delivery up to date value to the customer.

2.2.2 Manifesto for agile software development.

On February 11-13, 2001, at The Lodge at Snowbird ski resort in the Wasatch mountains of Utah, seventeen experienced and recognized software development "gurus", inventors and practitioners of different agile software development methods (Kent Beck et al) met to find common ground. What emerged was the Agile Software Development Manifesto [10]. This manifesto states the main values for Agile Development.

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

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

That is, while there is value in the items on the right, we value the items on the left more." [10].

2.2.3 Principles behind the agile manifesto

Usually Manifesto for Agile Software Development is mentioned together with 12 principles behind it. We suppose that these principles are valuable source for testing NAPDiP concept as an agile software development process. As well following these principles helps the authors to make NAPDiP more individual oriented then its “non agile” counterparts. The authors do not question validity or sufficiency of these principles and accept them as it is.

Principles behind the Agile Manifesto are [10]:

1. *Satisfy the customer:*

   Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. *Welcome changing requirements:*

   Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. *Deliver working software frequently:*

   Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. *Motivate individuals:*

   Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
5. *Interact frequently with stakeholders:*

   Business people and developers must work together daily throughout the project.
6. *Communicate face to face:*

   The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. *Measure by working software:*

   Working software is the primary measure of progress.
8. *Maintain constant pace:*

   Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. *Sustain technical excellence and good design:*

   Continuous attention to technical excellence and good design enhances agility.
10. *Keep it simple:*

    Simplicity, the art of maximizing the amount of work not done, is essential.
11. *Empower self-organizing teams:*

    The best architectures, requirements, and designs emerge from self-organizing teams.
12. *Reflect and adjust continuously:*

    At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

2.2.4 Agile development processes

Overall, software development is a highly complex field with enormous quantity of variables involved in the system and all this systems are not perfect as imperfect humans are involved. They are heavily dependent on accuracy of the person who specifies the requirements, professional skills and previous experience of developer and a lot of other factors [12]. If we are going to build a house we can rely on physical laws. Software development has no laws or rules defined by its very nature. Another thing is when we build software we usually use other software systems, which are already developed and may contain their own bugs and (badly written) code artifacts. So, the conclusion is that a complicated piece of software cannot be built without changes appearing during its development (changes of the requirements or findings of the bugs in "building blocks" which prevent a developer from usage of the piece of software). It requires flexibility from the development process. *"It is therefore fair to say that software development is more akin to new product research and development than it is to assembly-line style manufacturing. Software development is innovation, discovery, and artistry; each foray into a development project presents new and difficult challenges that cannot be overcome with one-size-fits-all, cookie cutter solutions."* [11].

So, the next step after the waterfall model (which consist of consequent phases with strict order and predefined state in the end of each phase) [9] was an iterative development. With this approach the development lifecycle is cut up into increments (or iterations), and every iteration includes all stages of the traditional "waterfall" model. This solved problem of scope changing which are coming late in the process or development. This approach to software development allows for multiple "passes", over a project lifecycle to properly address complexities and risk factors.

The common feature of the agile development processes is that they break tasks into small increments. As well agile methods usually do not include long-term planning. Increments (or iterations) in these processes are short (usually 1-4 weeks). Every increment includes planning, requirement analysis, design, coding and testing. Usually the whole team (including designers, managers, developers, and other roles) is available during a whole software development cycle until the moment when a working product is shown to the stakeholders or customers. This helps to adapt the software to the changes coming from the customer requirements faster and generally adds flexibility to the process. Iteration may not add enough functionality to provide a ready-to-shelf product, but at the end of each iteration the customer is supposed to be able to see the working piece of software with limited functionality. Obviously at the end of the last iteration customer supposed to get full version of the software.

Usually agile software development process assumes self-organization of the team without clear hierarchy (corporate hierarchy or corporate roles) during the development itself. In the agile concept there is no such thing as “task assigned to the developer”. Every developer selects tasks for himself from the project scope according to his own skills and estimations. Usually each team member decides individually how to fulfill the requirements stated by the feature he works with.

Usually agile methods assume face-to-face communication and comments made inside the code instead of written documents. Furthermore, it assumes that the whole team is situated in the same location (but unfortunately this is not possible in the international projects; just the part of the team situated in one city can exploit this feature). There is an opinion that phone call conversations and daily videoconferences can be used instead, but in practice the difference is significant [13].

Typically agile teams are small enough to fit one office and to maintain effective communication, usually agile team includes 5-9 people. Larger development is considered as parted for several agile teams, which are working under centered coordination from common manager [13].

The common thing for agile processes is a customer representative role in the team. He is appointed by the stakeholders to represent the needs and requirements of the stakeholder. He should be available to the team during the development process and should be able to answer the mid-iteration questions. At the end of every increment, the stakeholders review the progress [13].

3 Purpose

There are many best practices and processes available for software development. Some of them are not suitable for agile software development and some of them don't fit to distributed projects. Therefore, a set of best practices considered to be a necessity in a distributed project development running in agile way. "The new agile process for distributed projects" is such approach or a way of work with global outsourcing. This research was initiated by its authors and an IT service company in northern Europe. The process provides benefits to the domain knowledge spreading and proposes a good branching and release management. It also provides a set of best practices, helping to link design, test approaches and tools, which help to maintain software architecture. The process emphasizes time to market and product quality on an acceptable level. Besides this the approach gives the opportunity for evaluation and improvement.

The purpose of this report is to conduct an extensive research on NAPDiP, explore approaches according to agile development models and manifesto, identify the issues and challenges of distributed software development projects especially in communication because it is vital for success of a global project. Furthermore, researchers aim to finding solutions and additional useful practices, redefining of some proposed practices for improvement of NAPDiP and appropriate management of distributed software development. The final goal is to share the knowledge and apply the proposed modifications and practices to the process and used by globally distributed software development projects.

4 Research Approach

In this chapter we will provide to the reader clear and systemized information about methodology used and the reasons for selection of this methodology. As well alternative approaches will be mentioned and supported by the reasons why they would not be used.

According to [14] there are three types of purpose of an academic study: exploratory, descriptive, or explanatory.

Exploratory studies are practical if you wish to clarify your understanding of a problem according to Saunders et al [15]. They state exploratory studies as a method of finding out “what is happening; to seek new insights; to ask questions and to access phenomena in a new light”

Descriptive studies suite best for description of a phenomenon such as a process or an event. As well, this way of research is appropriate in the case of the clearly defined problem, but a researcher mainly aims the process itself instead of focusing on connections between symptoms and causes.

Explanatory studies according to Saunders et al [15]; best fit if the researcher wants to explore dependencies and correlations between different sub processes and data in the research object. The main thing in this sort of study is to examine a given problem in order to explain dependencies between initial data and result.

This paper aims to combine these three approaches. The study is explanatory because we are going to find out dependencies between different aspects and practices of NAPDiP process. On the other hand it is descriptive as we are interested in studying the process itself, finding out new practices, and approaches for modifying the existing process. Furthermore, the research is an exploratory study by its nature as we are interested in understanding and clarifying what happens when NAPDiP is applied to the international project on the current stage.

According to another gradation, described for example by Bockmon [16] the research is mostly qualitative as the intention is not to measure something but to "study things in their natural setting, attempting to make sense of, or interpret, phenomena in terms of the meanings people bring to them". The only quantitative part of the research is trying to investigate and measure the success of software international project running in agile way.

Another thing, pointing to the qualitative methodology is that this paper does not include the questions like “How many hours should be spent for training NAPDiP users?” or “How many features should be implemented by a developer to maintain the healthy pace of work?" (for these questions quantitative methods have to be applied), our intention is to find the solutions of the problems like "What is the best way to convince developers that writing unit tests is great solution?", "Which best practices can be applied to improve communication between people working in the different parts of the world with the same project?" etc.

According to [17] there are three main methods of qualitative research. These are

- Participant observation—where the researcher also occupies a role or part in the setting, in addition to observing.

- In depth interviews— face to face conversation, to explore the issues or topics in detail. Any preset questions are not used, but is shaped by a defined set of topics.

- Focus group— a method of group interview, group interaction is explicitly includes and uses to generate data.

Greenhalgh and Taylor in their paper [18] add two more methods to the three mentioned above.

- Documents—study of documentary accounts of events, such as meetings

- Passive observation—systematic watching of behavior and talk in natural occurring settings and even more.

- Diary methods - the researcher or subject keeps a personal account of daily events, feelings, discussions, interactions etc.

- Role-play and simulation - Participants may be asked to play a role, or may be asked to observe role-play, after which they are asked to rate behavior, report feelings, and predict further events.

- Case-study - This is an in-depth study of just one person, group or event. This technique is simply a description of individuals.

Unfortunately not all of them are appropriate. It is not possible for researchers to be a part of the process which they study if it requires from them to be the part of the team of an international project which uses NAPDiP process. As it requires appropriate software development and other special skills. So participant observation is not an acceptable method on this level of research. As well right now there is no currently running NAPDiP project and it causes inability to use passive observation. As the aim is to study the process – there is no interest in personal qualities of participant of the process and as a result there is no possibility to use diary methods. Also the goal of the research is not to study personal relations between developers, as well project is a very time consuming action, so role-play and simulation are not appropriate either. So, the main methods of research are documents studying and in depth interviews. Focus groups research will be avoided as it requires significant expanses to the company.

Literature reviews and studies are significant part of the research as well because improvements of the NAPDiP process can be based on already existing techniques and practices which are used in other processes. For example the experience collected by usage of XP or Scrum frameworks should not be underestimated. This experience would be applied for the research; aiming finding strong points of these frameworks applied to international project and adaptation of them to NAPDiP.

Another very important issue which makes significant influence to the research is the quantity of real projects running with NAPDiP. Their number is extremely limited and as a result issues appearing in multinational projects would be identified. Due to this, the research is based on either projects running with NAPDiP or projects running with Scrum, RUP and other processes. The interviews from multinational projects which use different processes are collected and analyzed.

4.1 Interview

Interview is a widely used method for collecting qualitative research data because "it is perceived as 'talking' and talking is natural" [19]. This research is focused on interviewing of participants in different projects that run internationally. Main focus of the interviews is to get in depth understanding of the projects running in agile manner and to find out the issues or difficulties within the project and focus on finding suitable solutions for that.

Interview strategies are important for pulling out reliable, comparable and qualitative data. From different interview strategies the focus is on semi-structured, open-ended interview technique which is advantages of high validity by talking detail & depth and also complex questions & issues can be discussed or clarified. Semi-structured interview allows both researcher and interviewer to be prepared ahead of time and appear competent during the interview [20].

In this research project the aim is to interview different key personnel from different multinational projects. The researchers had the intention to interview the project managers, team leaders, developers and configuration managers for understanding and finding the issues or difficulties that arise during the project run time. The main focus of the interview questionnaires is about the development process and its implementation, as it is key component for this research project.

Verification and validation of data received from the interviews are important for further analysis and finding the issues. Hitchcock and Hughes [19] suggest using triangulation and re-interviewing for validation of interview data, where triangulation is comparing data from at least two sources. Triangulation can be done by comparing interview data with project data or interviewing same person again by asking some of same questions or interviewing other person with the same questionnaires. Re-interviewing has different strategies for validation of data and requires significant amount of time to follow the process. Sometimes it is not possible to interview the same person several times.

Interviewing is not just collecting the data. It aims to analyses of the gathered data and finding of tangible issues which arises during the process and its application to the project. The information is supposed to be analyzed by comparing with project descriptions and aims. Findings from the experienced personnel are supposed to help researchers to discover best solutions.

5 “The new agile process for distributed projects (NAPDiP)”

5.1 Overview

“The new agile process for distributed projects (NAPDiP)” is an approach to global delivery model in agile way. To fulfill the agile manifesto, NAPDiP has incorporated many of its best practices from different agile and iterative development processes such as ROPES, SCRUM, RUP, and XP [13], [30]-[35]. NAPDiP involves the customer in the entire process and deliver quick prototype for evaluation and an effective alternative way of monitoring progress. NAPDiP delivers adequate documentation for each implemented use cases and relevant test cases for those use cases. It preserves the framework and prevents the violation of architecture. NAPDiP uses spiral model which is used in the ROPES process but the concept is different. Its micro spirals concentrating on features next to be integrated and the project team together with the customer will decide what features to be integrated next. In NAPDiP the design, implementation and test phases are considered as iterative and the next micro spiral will not starts until the test phase has been finished and a stable runnable prototype is available. An evaluation of previous increment done at the end of each micro spiral that reduce the risks and ensure the process is working well.

According to the authors of “the new agile process for distributed projects”, the core ingredients of this process are:

- NAPDiP is an agile approach to global delivery.
- NAPDiP is model and test driven.
- NAPDiP delivers daily runnable prototypes for the customer to download and try out.
- NAPDiP delivers adequate documentation for each implemented use case.
- NAPDiP delivers relevant test cases to each use case.
- NAPDiP preserves the framework and prevents architectural violation.
- NAPDiP continuously integrates new features on a stable base.

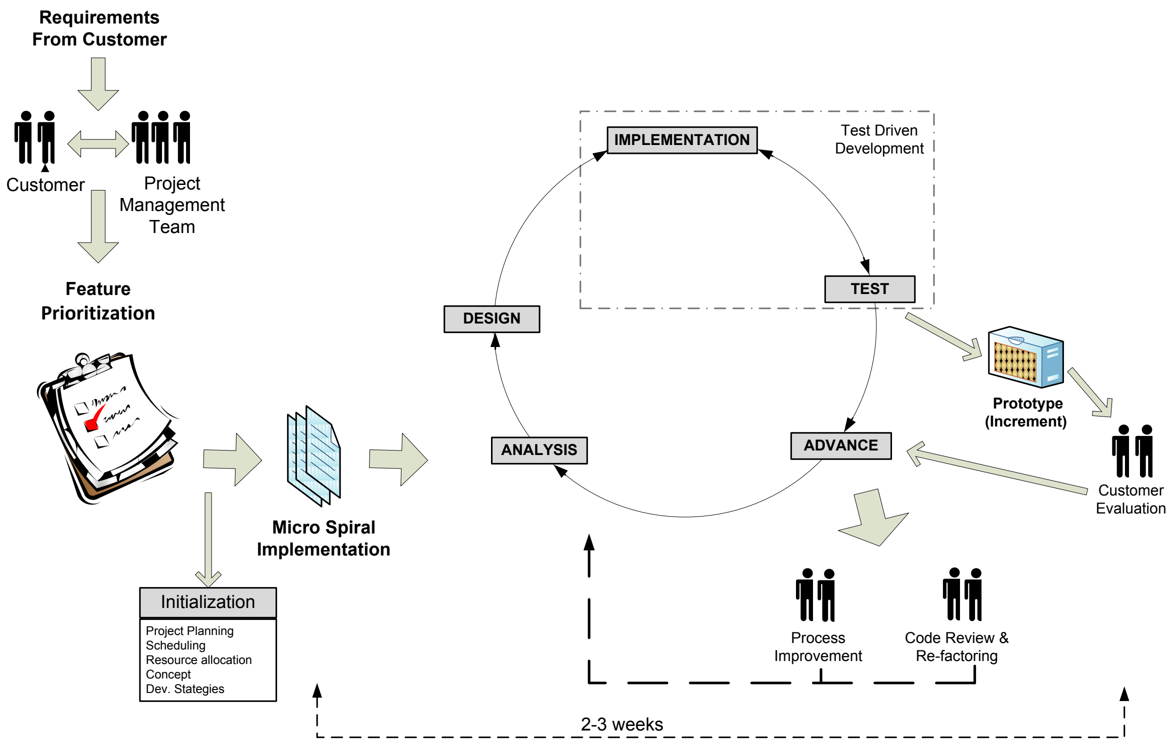

Figure 1: "The new agile process for distributed projects" working framework

5.2 Project responsibilities

Due to the large variations in sizes of the development teams the composition of the project roles can vary. The most common case is that the same person can represent several project roles. According to the authors of NAPDiP, it proclaims that one shall keep the number of roles to a minimum level. Unlike SCRUM, NAPDiP needs a specific role definition. The core idea of the approach is that each individual takes responsibility for a given activity. It also suggests that the responsible subject gives the project management team, time estimations for their assigned activities. Following set of roles involved in the process:

- ***Project manager***, this is the person responsible for monitoring the deliveries and to handle resources. The project manager shall also arrange feedback meetings with the stakeholders. As well this person is involved in the establishment of the time plan, being in communication with both the development team and the customer.

- ***Architect*** is the person responsible for establishment of one or more architectural rules for software providing good performance, testability, scalability, maintainability and modularity. An architect shall also interact with the test & quality manager when it comes to questions concerning the testability. Moreover, the architects' most important task is to form a strategy, which prevents the executive programmers to violate the architectural rules.

- ***The Test Designer*** role is responsible for defining the test approach, division of tasks related to testing and ensuring its successful implementation. The role involves identifying the appropriate methodologies, tools for implementation of the required tests [21].

- ***The Feature Designer*** is person who is given the very specific task of conceptualizing and designing a component (or a group of components) for a system. Normally, this includes features, modes, or other parts of software. They normally report to the architect.

- ***Test and Quality Manager*** incorporates a key role, with responsibility for the quality planning process and monitoring the implementation of the relevant quality standards. Another meaning of the position of test manager is this is the person who ensures the timely delivery of all testing activities and associated data in accordance with the prescribed quality standards.

- ***Executive programmers*** are persons responsible for development of the software, code writing. As well their responsibility includes unit test writing and performing.

- ***The Configuration Manager*** role is responsible for configuration of the system and environment to the product development team. The CM function supports the product development activity so that developers and testers have appropriate workspaces to build and their code. The configuration manager role also has to investigate and analyze product review, and change and defect tracking activities [22].

- ***Delivery manager*** is the person responsible for insurance that the deliveries are stable and don't contain any "show stoppers".

- ***Requirements Manager's*** role is to elicit, analyze, specify, and validate the requirements received from the customer.

- ***Risk Manager*** is the person responsible for evaluation and minimization of the risks taken by the team during the development.

- ***Stakeholder*** is an integral part of a project. He is the end-users or client, the person from whom requirements will be drawn, the person who will influence the design and, ultimately, the person who will reap the benefits of completed project.

5.3 Project Initialization

During project initialization phase of "the new agile process for distributed projects" requirements from the customer/stakeholders are captured through the project manager who is an initial contact person for customers. At the very first initial project meeting several activities are conducted: project planning and scheduling, allocation of resources, development strategies including development plan, configuration management, and development of frame base.

In this initial phase of the project, stakeholders and the project management team prioritize the feature lists which are implemented in each micro spirals and integrated to the system. Project manager, software architect, test and quality manager, feature designer, configuration manager and executive programmers are involved in this phase. NAPDiP aims to offshore project information sharing and cross site education.

NAPDiP implements two or three features from the priority list in each micro spiral which is duration of 2-3 weeks. During this short micro spiral detailed analysis, design, implementation, test are done for each feature. Quick prototype of the implemented features is delivered for customer evaluation at the end of each micro spiral.

5.4 The life cycle of "the new agile process for distributed projects (NAPDiP)"

According to the authors of "the new agile process for distributed projects", this process uses a similar spiral model as ROPES [34] where each of iterations starts with an advance party. Unlike ROPES, NAPDiP doesn't have the same concept of focus. It has a focus concept but differs from ROPES [34]. ROPES divide the process in several micro cycles where one or several micro cycles focus on one domain, for example "Focus on key concepts" or "Focus on Optimization". However, NAPDiP has micro spirals focusing on features next to be integrated. The project management and developers together with the customer agree upon the features to be integrated next.

Domain analysis is not the part of NAPDiP so this activity is not discussed in the report.

5.4.1 Analysis phase

During the process each of iterations starts from analysis phase. This phase includes requirement analysis, architecture analysis and object analysis.

In the requirement analysis architect, delivery, configuration, quality and test managers are involved. Testing is very important to infrastructure during the software development.

To regulate the time plan of the development process managers need to have defined test plan. Others modify the formulations of requirements, specified by customer, transform "messy" customer wishes into the requirement document, understandable to the designers and the developers. For example customer states that administrator should be able to add the new user into the system quickly. During requirement analysis phase this should be reformulated to "an administrator should be able to add the new user into the system in three mouse clicks". At the end of requirement analysis phase, update of the requirement analysis document is delivered.

The second part of the analysis is Architecture analysis. Obviously architects are involved. It can be hardware architect or chief software architect with test manager, according to the type of the project. During this period architect creates components and defines relations between them. In the end of this phase software architecture document is delivered.

The last part of the analysis phase is object analysis. Test managers and software architects are involved. Main objects are specified and their relations are stated, as well packages and namespaces are defined. The test plan is corrected according to the result of this phase. Architecture analysis document is updated according to the results of this phase.

5.4.2 Design phase

After the analysis is completed the process goes to the next stage, design phase. It starts from architectural design part. The architect and feature designer collaborate here. Architect specifies requirements to the designer and the designer provides algorithms and practical solutions to the problems stated by the architect, here attributes of the projects and methods are specified. "The new agile process for distributed projects" encourages the usage of UML tools for modeling of the program structure, classes and interfaces are defined here. The result of this part is stated collaboration rules between different parts of the software and general structure of the program. The results are reflected in update of the architecture document. When the architecture design is completed it goes back to the architect for approval where "The '4+1' view model of software architecture" is applied [23]. It follows this micro loop during the whole architectural design part.

Architectural design is followed by object design. In this part only the feature designer is involved. He generates the code, consisting of empty classes and interfaces on this stage according to results of the previous phases. The result of this part should be reflected in the software architecture document update.

Next part of Design phase is impact diagnostics. During this phase designer analyze how the newly introduced feature would interact with other parts of software, what impact it will make to the other components.

After Impact diagnostics process goes to test design. Test manager and feature designer are involved. Test plans are updated; primary tests are implemented during that phase. As well interfaces for test are delivered during this stage.

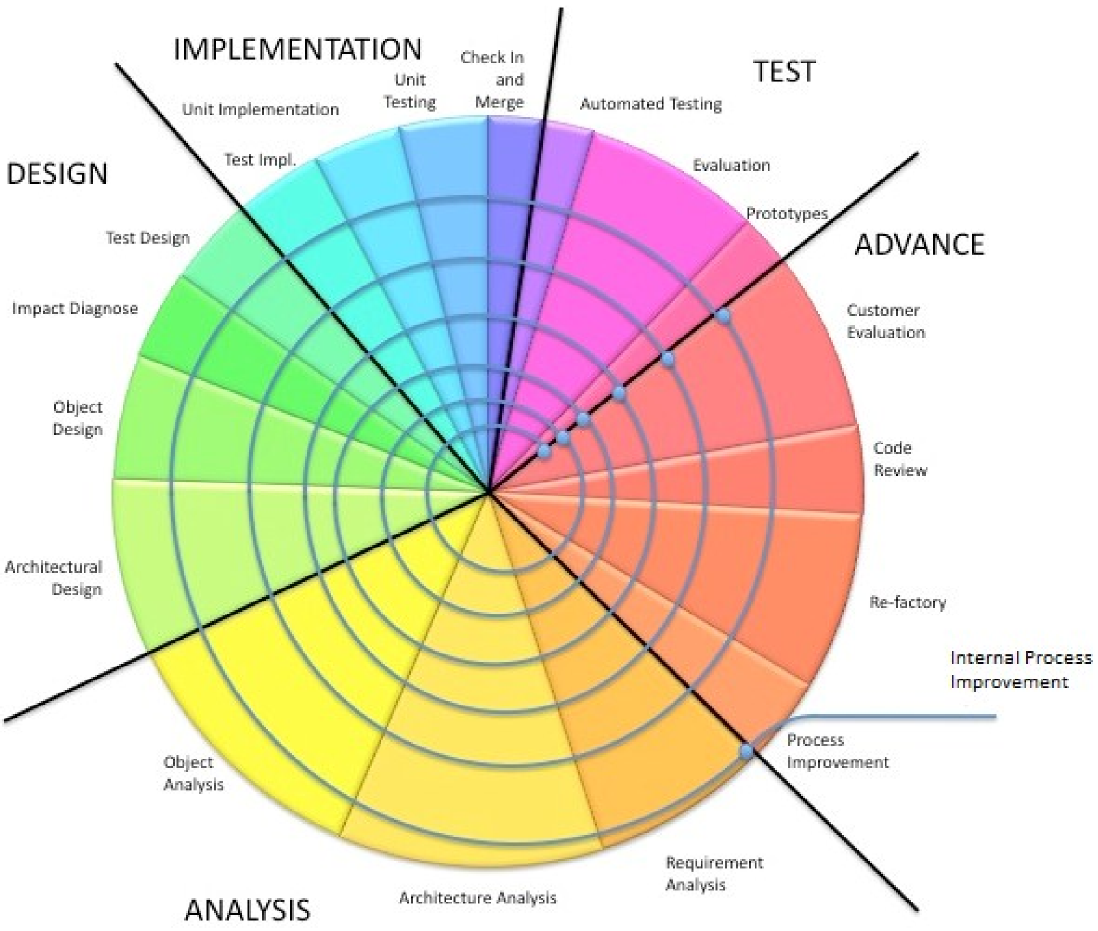

Figure 2: "The new agile process for distributed projects" Process Spiral

5.4.3 Implementation

The implementation phase provides concrete solutions to the features. Those were determined in analysis phase and designed using different modeling tools during the design phase. Here developers get the interfaces for their implementation; however they are not allowed to start the implementation before completion of the analysis and the design phase. It is very important to finish the test implementation and test cases before the unit implementation. During this phase designers and implementers are encouraged to work closely. It is preferable the same person plays the role both designer and implementer. In

NAPDiP every developer is encouraged to write tests before implementation. After unit implementation the unit testing and check in & marge activities are performed. The practice of test driven development is applied during the implementation phase. Mostly developers/programmers and designers are involved in this phase either offsite or onsite.

5.4.4 Test

The test phase assures that the prototype is already designed and implemented correctly and no violation is made to the existing architecture. This phase starts when all developers commit their parts of code into the repository. "The new agile process for distributed projects" encourages usage of different automated testing tools. It states that tests should be automated. Test results are evaluated and sent back to re-implementation if any tests fail. It follows a micro loop until the automated tests are passed. At the end of this phase prototype of implemented feature should be available for customer evaluation, which takes part on next advance phase. In NAPDiP design, implementation and test phases are considered as iterative and next phase (ADVANCE) would not start until test phase would be finished and a stable runnable prototype would be available.

5.4.5 Advance

In the advance phase customer is invited for evaluation. There he gives feedback on existing features by downloading the prototype which was finished at the end of test phase. Discovered bugs and issues are added and corrected during the next iteration.

Code reviews are performed during this phase as well. At this stage of NAPDiP the architect evaluates the code and notifies the designer if it needs to be changed. As well he checks if the code violates the architecture, initiates the re-factoring procedure if it is needed.

During the re-factory part of advance phase last document updates should be done according to the changes made during development, tests and code review activities. The last part of the iteration is process improvement, where all members of the team are involved. They give their feedback and experience from iteration to take it into account during the next one. The bugs, issues in the code review and refactoring of this phase are transferred to the next increment.

5.5 Best practices of "the new agile process for distributed projects (NAPDiP)"

- ***Continuous Integration***

According to this software development practice members of a team are obliged to integrate their pieces of code into repository. Usually every team member should commit something every day. Each committed piece of code is verified by an automated build (including test) on server. This procedure helps to detect integration errors as quickly as possible. It was proved by practical experience of many software development teams that this approach leads to reduction of integration problems and as a result the team is able to develop cohesive software more rapidly [24].

- ***Configuration and Release Management***

This practice assumes the presence of configuration manager and release manager roles. It allows the team (especially its management part) to deal with security features, assurances through control of changes made to hardware, software, firmware, documentation, test, test fixtures, and test documentation throughout the life cycle of an information system. It provides an opportunity to react on the flow changes faster.

- ***Test Driven Design***

This practice combines test-first development which assumes writing a test before writing just enough production code to fulfill that test and refactoring. In NAPDiP programmers are encourages to write test before implementation of code. It provides better structured code which is easy to test automatically [25], [26].

- ***Time to Market improvement***

Applying this principle NAPDiP reaches high speed of development according to changeable market intensions.

- ***Interface steering***

According to this practice software designers should define formal and verifiable interfaces for the developers; they should include preconditions, post conditions and invariants for the methods. In NAPDiP case it allows on-site part of the team to get an assurance that the code, gained from offshore, will be properly structured and would be easily maintained [27].

- ***Prototyping***

Prototyping provides the delivery of extremely limited and incomplete version of the software early in the development process. Prototype is a kind of a skeleton. It gives an opportunity for the customer to track the process easier [28].

- ***Self-Improvement, Waste Reduction***

In the advance phase the flow of the project is evaluated. It allows the team to improve and customise the process according to their current needs. As well the team tracks all steps of development and uses its observations for waste reduction.

- ***Re-factoring***

Re-factoring is the activity which includes changing a source code without modifying its external functional behavior. The intention of this activity is to improve some of the non-functional attributes of the software. As an outcome, team gets improved code readability and internal architecture which results in improved maintainability of the source code and extensibility.

- ***Concurrent Activities***

This way of work allows the team to analysis, design, write the code, test and improve the process in different phases at the same time. During each phase team members are occupied according their own responsibilities and it allows application of this practice.

- ***Automation***

Applying this practice NAPDiP teams use different modeling tools according to technical requirements specified during design phase. Following Test Driven Development, tests are also automated. In NAPDiP building prototypes is also automated after success of test in each cycle.

- ***Multi-channel Communication***

Multi-channel communication practice incorporates messages into different types of data representation delivered via two or more media channels, including print, e-mail, Web (personalized URLs or Web microsites), and text messaging. It can be documents, notes, phone talks, etc. The real value in multi-channel communications campaigns is the use of personalization for a more targeted, relevant, one-to-one approach that engages all team members in a dialogue. The usage of special software tools offers to the team to feel itself as a whole putting the team members closer to each other.

- ***Proof of Excellence***

This practice allows onsite staff to control the successes of offshore staff to ensure that a customer would get desired quality of the product. Usually it is taken the form of code and design reviews, done by onsite part of the team. This activity is usually performed at the end of every phase during the iteration and involves participation from both the offshore part of the team as well as from onsite part.

- ***Incremental Design***

In traditional software development the design phase is conducted in the beginning of the project, and as a rule it takes quite a long no-coding or little-coding time. At the end of the phase, the design is considered being fixed. However attempts to fix the design well in advance often lead to the wrong assumption and sub-optimal solutions. This practice allows the team to adjust the design according to on-going process of changing requirements [29].

- ***Model Driven Design***

This software design approach applied to the development of software systems provides a set of guidelines for the structuring of specifications, which are expressed as models. Afterwards these models are used in engineering of software systems.

- ***Use of Common Frameworks***

Use of common architectural frameworks allows NAPDiP project teams to work in distributed environment with highest level of quality within limited time scope. It improves maintainability of the source code and the architecture.

- ***Time Boxing***

Time boxing is a widespread time management technique used in planning projects (typically for software development). According to this practice the schedule is divided into a number of separate time periods (time boxes, normally two to six weeks long), with each part having its own deliverables and deadlines [36].

- ***Stakeholder involvement***

Active stakeholder participation is driving force of agile development. In "the new agile process for distributed projects" customer is involved in planning activities and evaluation of prototypes during each of iteration. Teams are allowed to contact with stakeholders if required. Customers' evaluations are analyzed during the advance phase, each of the iteration. This is one of the sources for the improvement in the next iteration.

Table 1: List of best practices and their benefits

Best Practice	Benefit
Continuous Integration	Early identification of development problems, ease of progress tracking.
Configuration and Release Management	An opportunity to react on the flow changes faster.
Test Driven Design	Better structured code which is easy to test automatically.
Time to Market improvement	Speed up the development according to changeable market intensions.
Interface steering	Increase the quality of the code, code is more structured.

Prototyping	Gives opportunity to the customer to track the process easier.
Self-Improvement, Waste Reduction	Allows the team to improve and customise the process according to their current needs.
Re-factoring	Improves maintainability of the source code and extensibility.
Concurrent Activities	Allows the team to write the code, test and improve the process at the same time.
Automation	Reduces time spent for testing.
Multi-channel Communication	Offers to the team to feel itself as a whole putting team members closer to each other.
Proof of Excellence	Allows onside stuff to control the successes of offshore stuff to ensure that a customer would get desired quality of the product.
Incremental Design	Allows the team to adjust the design according to on-going process of requirement changes.
Use of Common Frameworks	Improves maintainability of the source code and the architecture.
Time Boxing	Simplify the time management during the project.
Stakeholder involvement	Speed up the development process.

6 Analysis and Discussion

Implementation case study:

Software projects, which used "the new agile process for distributed projects", are an important source of research. This process was applied in one software development project. The project was distributed between two countries, onsite part of the team was located in northern Europe and offshore part was located in south Asia. There were some identified challenges beside the technical issues, for example cultural differences and time zone differences.

NAPDiP was applied to development of a software application. This application was supposed being able to store retail parts, handle customers and systemize users. This application was considered as a user intensive system and therefore usability was one of the most important requirements. It was also desired being able to extend the system in a simple way. Therefore, two other quality requirements were added, modularity and scalability. Finally, to verify the quality and that the architectural rules were followed, the testability was considered as a very important quality attribute. The application requirements would also help the process strategy to fulfill its purpose by:

- Using a global delivery where it is possible.
- Guaranteeing that no architectural rules are broken and thereby have a product with good quality.

Project initialization started with a resource allocation, management team in northern Europe picked the adequate competence from the offshore. The selection process started with curriculum vitae (CV) reviews and thereafter an interview was conducted with the individuals of interest. This was considered as enough effort. The project plan was produced and a configuration management strategy was also settled and presented in the project plan. Infrastructure of the project is shown in the Figure 3. The project plan assigned the roles and responsibilities to each project participant. The roles were adapted as described earlier. Since the offshore site desired to take more responsibility for the design, the feature design team was located in offshore site. A plan for evaluation of the process was also scheduled at the second party/spiral. Another issue of importance concerned the way information should be shared, therefore a set of templates was defined.

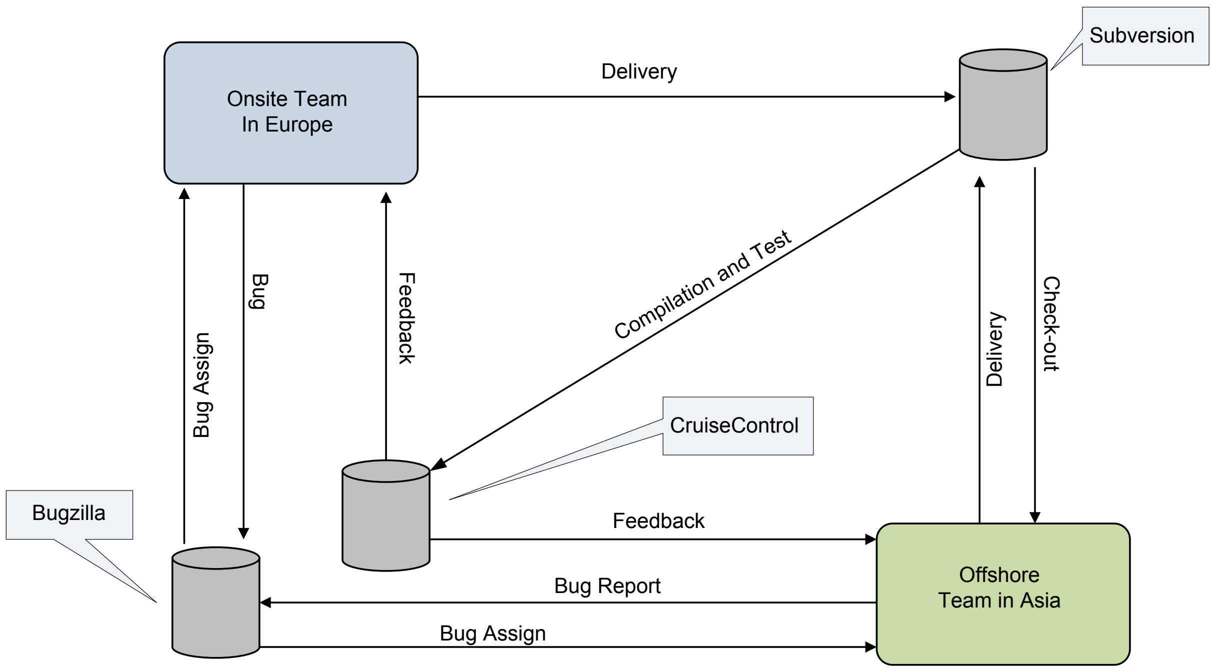

Figure 3: An example of infrastructure of "the new agile process for distributed projects (NAPDiP)"

For information sharing several teleconferences were held at the beginning of the project. The Management team strived to give as much information of the process strategy as possible. They also wanted to put focus on the most important quality requirements. Another activity inspired by SCRUM was that the project should have meetings on a daily basis at top level positions. The daily meetings were supposed to give an opportunity to share issues, report progress, report the daily workload and give estimation on future time expectancy. A deep architectural investigation of frameworks was also initiated at the party stage. It was considered necessary to have a framework as a base for the service oriented purpose. The framework should also make it easier to deliver self-testing components and ease the maintainability, scalability and usability. The process encouraged that the "architectural rules" were supposed to be stated as early as possible to move the focus from architectural issues to test strategy, features and functions. This strategy could also help to ensure that no rules were broken in the final delivery.

In the first cycle of the project requirements were captured by interviews. The stakeholder with domain knowledge addressed the system and the interview team asked questions if anything was ambiguous. There was also a seminar held for user interface requirements where the customer explained the desired view of the system. These requirements were compiled and sent to the feature designers at offshore site. From a management perspective the response expected from the offshore site was supposed to be a detailed design and a suggestion for modifications to the framework. A response with a detailed design and suggestions from the feature design team was submitted to the architecture team, as it was expected. In the first feature requirement specification the connection between UI's and data was omitted. This caused several misunderstandings and many mails were sent between the sites clarifying how data should be connected to the UI. The whole idea with interface steering had not been grasped at the offshore site. Changing an interface would have forced the executive programmers to re-implement the declared method in all the realizations of the interface. This fact was not noticed at the beginning of the design phase. The issues were solved and the architectural rules were clarified. The architecture team therefore approved the second design attempt.

The process required the test cases to be delivered with the design. This test driven approach ensures that the new features delivered can be tested properly and eases the ability to monitor the daily work. Review of the test cases would also reveal if any business logical or architectural principle had been broken. In the first design attempt the offshore site did not deliver any test cases. Since no tests were delivered, the test phase was omitted. Unfortunately, the code did not compile and the feedback system started to send out mails to all involved subjects. If no control were attached to the committed source code, a build would fail. In the strategy used by the process, error is addressed immediately. Programmers from the onsite and offshore team made the corrections and in a few hours the trunk was considered to be stable again.

After an analysis, several risks were identified during the first cycle. One of the risks was that the complexity of the framework would be hard to communicate and it would lead to a high time consumption. Another risk, which is very common, is the language barrier. Language issues could cause misunderstandings and consume more time than expected. Another anticipated risk with high probability was that the core ideas of the process could be poorly communicated.

In the second cycle, the quality manager reviewed the source code for the two features and feedback was sent to the offshore site. Corrections were made by the executive programmers and the stakeholders were invited to give feedback on a daily build. Due to a late modification of the build, the application crashed and the stakeholders were unable to perform the scenarios. This fact resulted in many reported bugs and a priority list for each bug. The stakeholder assigned a priority to each bug and a desired fix date. The problem could have been avoided if the delivery manager had picked a known stable and working release for feedbacks. In this case the stakeholders picked the latest available release, which wasn't working. This was the result from a communication failure to the stakeholders. Another communication failure seemed to be that the offshore site did not deliver any test cases with their design. Language problems could from time to time occur as an issue due to bad connections during the morning meetings. It was then decided to e-mail their questions as a precaution to avoid confusion. Finally, it was determined to integrate a more complex feature for the next increment.

The second design phase ran smoother than the first design phase. Still there were some violations to the architectural rules, which were quickly corrected. The sequential flow was also logically wrong and would have caused future errors. This was corrected by the architecture team and after three reworks from the feature team the design was approved. The management team wanted to see if a sequence diagram would be enough to communicate the design solutions. Another incident happened during this phase as well; the chief architect approved a faulty design. This was later discovered and a correction had to been communicated to the feature designers in offshore site. During the design phase, the offshore team also worked hard to complement the first features with sufficient test cases. It turned out that the test approach was unclear to the feature design team and they had some troubles to deliver the test cases. After some clarifications they managed to deliver sufficient test cases, but no test cases were delivered for the new feature. The omitted test cases were most likely the result of communication failures and poor training concerning the test approach.

The entire project has consumed more time than initially calculated and the integration took two more days than expected. The principle to check in every day was not followed by the feature designers. This was devastating from a progress perspective, since it was impossible to determine the current state from day to day. The first delivery did not compile and was corrected by the part of the onsite team. Unfortunately the next customer feedback meeting had been scheduled and since the delivery was delayed a delivery assurance was not possible. Since the deliverance only existed in the three latest builds, the stakeholders downloaded the latest build and reviewed the result.

In the feedback party, architect team reviewed the code and found no violations to the architectural rules, but they discovered several errors in the user interfaces and the action commands. All errors were reported to the error handling system. There were too many errors for the programmers to correct in time for the feedback meeting, due to the short time frame. The management team decided to have a feedback meeting even though there were known errors. Feedback from the stakeholder was everything but positive. Several actions did not work and the screen layout was not as the stakeholder had expected. Further, the stakeholder was not able to use the features from the first increment. A time frame estimation to correct those errors and settled by the onsite team.

During our research, several practices, roles, additional stages were found that can improve global development process if they would be applied properly in “the new agile process for distributed projects”. In this section we will enumerate them and provide analysis of their addition into NAPDiP.

(6.1) According to [1] it is impossible to organize proper development process if developers from the beginning don’t know which code conventions to use, if the business domain is unknown area for them or if they didn’t get the deep insight into the architecture.

In this case an enormous amount of time will be spent by the team just for learning the architecture, requirements, etc. On the other hand, properly organized knowledge transfer will allow the team to learn more from and about distant members and it will remove significant amount of questions about initial requirements, architecture. Knowledge transfer conducted at the beginning of the project will allow off-site to gain knowledge about the product and business domain. In the case of the off-site personnel being involved in the further development of an existing product, knowledge transfer activities should also convey information on working and communication practices, as well as on how to use development and testing environments and frameworks.

The most important knowledge transfer activity should be conducted on the early phases of a global software engineering project, but it should be ongoing activity that prevents stuck from the beginning of development. on one hand, it might seem logical to do most if not all knowledge transfer right up front, but it is important to keep in mind that, for any even moderately complex product, it is unlikely that a short up-front training can convey all the necessary information to get new people up to speed.

According to [1] Initial knowledge transfer is preferably undertaken in co-located hands-on training sessions combining theoretical lecture-style teaching with a hands-on learning-by-doing approach. Working hands-on, preferably on real but simple development tasks, is more motivating for the people learning the system than purely theoretical teaching would be. Doing knowledge transfer face-to-face, possibly involving parts of both the on-site and off-site teams, provides the project team with possibilities to socialize and engage in team building activities. Allowing the project members to build trust and personal relationships with each other will be very beneficial, as it will enable more active communication between the sites once team members are distributed.

So, we propose to include knowledge transfer activity at the end of a design phase for developers and at the end of an analysis phase for designers.

The most important is transfer the knowledge between every site. This should be done by having designers moved for a week or two to the onsite and study the business domain. After this activity they should be able to deliver this knowledge to all team members.

(6.2) The second point to mention is an easily traceable hierarchy. We propose to make a stroke on this practice because sometimes according to [1] remote developers can not define the person responsible for a certain part of the process or project. That leads to incompetent decisions and answers given by “wrong” persons. And as a result this causes delays in deliveries, broken architecture and other unwanted consequences. The situation when the hierarchy is obvious and every team member can easily find out the person to whom he can direct his question allows the project members to work more efficient. This is even more so in a distributed environment, as the transparency between sites is always limited. In order to efficiently plan the division of work and responsibilities, there should be a clear organizational structure that is clear for everyone involved in the project. This can be done, e.g. with the help of a project website, the use of social media, as well as by encouraging and supporting face-to-face meetings. Right now almost every more or less significant development activity has a set of document with the list of persons according to the features' responsibilities, the developer can always find out who his manager is and who manager of his manager is, etc. As well code repository tools like SVN allow developers to keep track of code updates. However if the offshore developer intends to clarify the requirement or he wants to go for vacation he could not find the person responsible for these activities onsite with ease. And this modification was proposed to avoid the issues caused by nontransparent hierarchy.

Consider the situation: a developer has a question according to business domain; first person to ask would be designer. But it is common case designer is not an expert in business domain. Next the developer would ask the team leader, manager, etc. trying to find the expert. Traceable hierarchy would help to annihilate waste in form of involvement additional persons.

(6.3) During the interview on one of our case study projects developers mentioned that it was extremely useful to have a customer representative (or proxy customer, the person directly communicating with the customer, this person should be able (either he knows the answer or sanctioned to give his point of view as customer's point) to answer the majority of questions appearing amongst developers) as a part of the team. At the same time this person was a kind of cultural liaison. So he acted as a link between two sites, leveling the cultural differences and breaking language barriers. As well he allowed separating customer from the development team as all communication between developers and customer was going through him. We will illustrate the usability of cultural liaison by the words from [1]:

("In one case project, an off-site team member was moved to the on-site team. This person had several contacts at the off-site location and knew most of the people there. Thus, whenever someone from off-site had a problem and did not know whom to contact at the on-site location, he would instead contact this liaison. The liaison was then able to work as a proxy between the sites as he knew most of the people from both sites. Another benefit of this liaison was that he was also able to work as a proxy in the distributed meetings between the two sites. The developers from offsite were usually unable to understand everything that was said in English and sometimes were even afraid of speaking in English. This cultural liaison was then able to work as an on-demand translator between the team members during the meetings.

"I think that during training they found that it might be useful for the project if they had someone here [from off-site] who knows the situation there [at off-site]. So one of my tasks is also to serve as a proxy between [off-site] and [on-site]."

– Cultural liaison

The third benefit was that the work at the on-site was becoming more transparent to the off-site team. The liaison was able to communicate to the off-site the true feeling of how the project was progressing at the on-site.” [1]).

(6.4) During our interviews and according to [30] we found that keeping developers involved in tasks division is an extremely useful practice. This practice is commonly used in most of agile development processes in one or another form. First of all during this process developer would get the overview of the task before he will get this task. The second benefit is that developer would be able to influence on the time which will be allocated on the task and it would take in account his skills and knowledge. The leader would have the possibility to allocate the tasks according to the available resources (e.g. senior and junior developers) and that would result significant time reduction. The third thing to mention is increasing the skill level of developers. Consider the situation. Junior developer receives the task, but according to his lack of skills he was allocated some extra time that he would spend on learning of new technology or pattern required by the task. And the next time when he would receive something similar a smaller amount of time would be required.

For now this practice would not be included into the process as the authors of the process are satisfied with the effectiveness of the task allocating method used nowadays:

“We do encourage learning among team members and spread the knowledge among several team members so we won’t be disrupted if someone should leave our team. There should never be a hero in the team that alone knows part of the design or implementation. We also distribute the responsibility of components to groups within the team so they will be “experts” dealing with issues concerning that particular component but there should never to one single person.” - Initiator of the NAPDiP.

(6.5) Another possible way to improve the process is adding the practice which would force the developers to communicate onsite part of the team directly, instead of communication via local team leader. It can take place due to lack of English knowledge or habit of solving all issues and asking all questions to the team leader. This adds additional link into communication chain and results increasing the number of misunderstandings between onsite and offshore parts of the team. In the same time direct communication either via e-mail or phone conference would give to the developer the ability to ask and ask once again until a clear answer is given for the question. As well it saves the time that the team leader spends to communicate with developer and to pass the word from developer to the person onside to whom this question was addressed. However, improper usage of this practice could cause inconveniences, mostly due to lack of common language. Problems caused by misunderstandings can be revealed later and a significant amount of time could be wasted. However on the other hand the easiest way to solve a problem is asking someone, sometimes offshore team (according our interviewees) uses the possibility of questioning the onsite team excessively. That lead to a waste of resources in form of time spent by onsite team answering unnecessary questions (e.g. onsite part of the team provides the information to offshore part of the team that offshore part can find itself).

(6.6) According to [13] and [1] distributed daily Scrum meetings are extremely useful practice for distributed projects. In these books lots of benefits of daily Scrums are mentioned. First of all they provide frequent possibilities to share information and coordinate work between distributed team members, at second daily meetings help to recognize possible problems on the early stages of development, at third they provide a possibility to create contacts between different sites of project teams, they encourage team members from different sites to communicate more actively, also facilitating off-line communication after the meetings. These meetings provide a perfect way for every member of the distributed team to get an overview of the project situation. It states: "our interviewees reported that it was easier to monitor the offshore situation than before." [1]. Moreover, daily Scrums help to identify different development and testing issues quickly, since it provides with daily monitoring from every member of the team. If during the meeting any problems or needs for one-to-one discussion are encountered the interested team members should set up separate meetings after the daily Scrums and continue discussions in smaller groups or one-to-one either by video-, or teleconference, chat or email. It states: "In all our case projects, daily Scrum meetings encouraged team members to communicate more also outside the meetings, which was seen as one of the greatest benefits of these meetings." [1].

However daily Scrums issue several challenges for the "not used to this practice" teams. The biggest challenge for international teams is the same as for co-located teams. This challenge is to understand what the correct amount of information to report in a Daily Scrum meeting is. Obviously it is hard to define even in a co-located project, but in a distributed project it is even more difficult. From the beginning of using this practice the team members do not know what the others find interesting or important. And usually the team needs to practice this with the help of their Scrum master, or in NAPDiP case team leader, who plays the role of scrum master in NAPDiP. According to [1], it stated "In one of our case projects, the daily Scrum meetings initially lasted only a few minutes, before the team members learned to discuss actively and in particular to be open about their impediments. The Scrum masters started to encourage everybody to talk and share more about their tasks and impediments. Thus, the teams ended up having 15-minute meetings that were found very useful by all the participants." The second challenge is cultural differences, because they may have a big impact on what people find appropriate to report in a daily Scrum meeting. According to [1] there are huge cultural differences in revealing impediments and discussing them in daily Scrum meetings. It stated that "For example, in Scandinavian cultures talking about impediments is much more natural than in Asian cultures. Moreover, when team members come from different companies, they are more likely to hide problems, in particular in the beginning of a project." Also there is a large difference between projects that had distributed daily meetings and projects with co-located daily meetings. According to Guide [1] most of the participants of the distributed meetings mentioned the benefits: increased transparency to the other site, getting a good overview of what was happening in the project and well-functioning and open communication across the sites. However, the participants of the site-specific, non-distributed meetings talked about problems: they did not have enough communication and contacts with the other site, nor did they know enough what was happening at the other site. So, applying to our case, the developers should have daily meetings with designer of their module to give him the possibility to correct his estimations, reallocate resources, keep track of progress, as well someone from onsite part of the team have to be involved, he would give vectors for development, explain expectations from the offshore part and will keep track of the progress.

In [1] next Daily Scrum tips are proposed:
"Provide a good infrastructure for daily Scrums. Meetings should be easy to set up and provide as rich communication facilities as possible: virtual reality systems or videoconferencing are the best. If unavailable, a good quality voice connection will do, perhaps augmented with web cameras. Use text-only meetings only as a last resort. Avoid asynchronous 'meetings'".

- Work actively with the team by practicing and discussing, to find the optimum type and amount of information to report in the daily Scrum meetings.
- Create an open atmosphere that makes it easy to raise problems and issues without fear.
- Encourage discussions in small groups or one-on-one after the daily Scrum meetings and arrange a technologically good infrastructure for these distributed discussions."

7 Result: Modifications

This section presents the summary of the findings and suggestions from section 6 (Analysis and Discussion) and describes “the new agile process for distributed projects”, modified according to suggested improvements, best practices and future proposals in different stages.

7.1 Best Practices

Knowledge Transfer: according to the section 6.1, it is extremely important activity which should be conducted at the early stages of the project. Considering the framework of NAPDiP the knowledge transfer activity should be conducted as initial activity at the project initialization phase and to be ongoing activity during the whole iteration. The most significant stages of it should be conducted at the end of analysis phase for designers and at the end of design phase for developers and testers.

Traceable Hierarchy: as a result of analysis presented in section 6.2, this practice allows project team members to contact with right person for desired information. It causes reduction of misunderstandings and other undesired consequences and as a result improves transparency between the sites. It can be achieved by usage of appropriate tools, like project website, communication/social media, etc. To follow the design by contract practice more precisely and efficiently this practice is introduced.

Team based task division: keeping developers involved into the task division activity is an extremely useful practice which is commonly used in agile processes. Every member of the team knows more about his own skills than team leader and as a result can estimate the time he would spend for the task more precise. It supposed to annihilate waste in the form of selection of wrong persons for inappropriate tasks. For now this practice is not supposed to be included in the process according to the authors of the process. According to this research different approaches of task allocation should be tested in future projects, which would help to find the effective way of task allocation.

Direct Communication: according to the section 6.5 this practice empowers offshore developers to communicate with onsite part of the project team directly, instead of communication via local team leader. This practice is supposed to minimize the quantity of misunderstandings between the sites and build up the team as a whole. This activity should be conducted during the entire process especially design, implementation and test phases. The research revealed the risk connected with application of this practice: sometimes offshore site uses this possibility extremely extensively, causing extra expenses to the customer.

Daily Meetings: according to the section 6.6 it is proposed to implement the daily meetings (like: daily scrum meeting) between distributed sites/teams as well as in each site/team. This should encourage team members from different sites to communicate more intensive. It supposed to improve the efficiency of each team and give possibility to developers to correct their estimation, to managers to reallocate resources and keep track of the progress by the whole team.

7.2 Additional Roles

Proxy Customer (Customer Representative): it is widespread role in agile development processes. This is a person maintaining the direct communication with customer and acting as the customer regarding to the development team. He should be able to answer the majority questions appearing among developers according to business domain, requirements, etc. According to section 6.3 it is proposed to introduce this additional role and involve it in the entire spiral. It is especially valuable during design and implementation phases. Either project manager or architect can be proxy customer at the same time.

Cultural liaison: cultural differences are one of the key challenges of globally distributed projects. Considering the section 6.3 it is proposed to introduce the role of cultural liaison in the process. This person is supposed to act as a link between two sites leveling cultural differences and breaking language barriers. Anyone from the project team can carry out this responsibility in the development process, but this person has to be familiar with traditions of both countries and to have large experience of working in international teams.

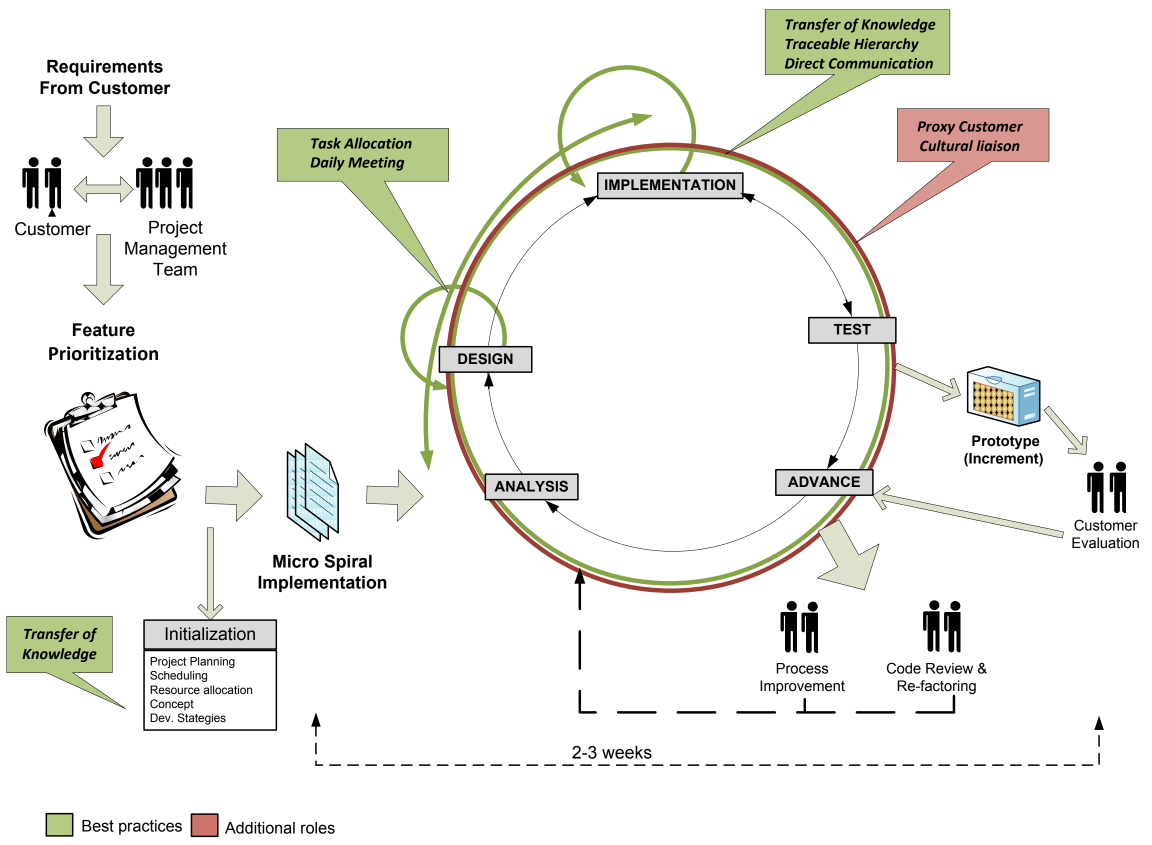

Figure 4: Suggested improvement and practices in different phases of "the new agile process for distributed projects".

Table 2: List of suggested best practices and their benefits

Best Practice	Benefit
Knowledge transfer	Allows the project members to build trust and personal relationships.
Traceable Hierarchy	Helps to annihilate waste in form of involvement additional persons.
Team based task division	The leader has the possibility to allocate the tasks according to the available resources.
Direct Communication	Lowers quantity of misunderstandings between sites.
Daily Meetings	Increased transparency to the other site, getting a good overview of what was happening in the project and well-functioning and open communication across the sites.

Table 3: List of suggested additional roles and their benefits

Role	Benefit
Cultural liaison	Levels cultural differences and breaks language barriers.
Proxy customer	Gives to the team precise view of customer needs.

8 Conclusion

The goal of this research was to systemize and describe the existing stage of “the new agile process for distributed projects” and propose suggestions for its modification. The first objective of the research was to systemize the practices, phases and other attributes of the process and to describe NAPDiP as precisely as possible. Based on the interviews with initiator of this NAPDiP process, presentation and project documentation of the process was described. The second objective of the study was to identify issues causing negative effect on the process. Comparison with other software development processes and analysis of the project experience revealed several leaks and weak points of the process. The third objective was actual improvement of the process. According to the detected weak points several improvements were proposed. They took different forms such as extra roles on the process, additional practices or activities which were not performed (or performed on insufficient level before).

As it was mentioned above the research resulted introduction of several best practices and additional roles. It was found that knowledge transfer as initial and ongoing activity speeds up the development process. Traceable hierarchy practice helps to avoid involvement of indifferent team members in communication within interested party. As well it assists to reduction of misunderstanding and to improvement of transparency between the sites. Direct communication empowers the developers to contact with onsite team for any unclear requirements without wasting time. Daily meetings encourage the developers to communicate actively with others and keep track of the progress.

It was found that additional roles like proxy customer, (person acting as the customer for the development team) and cultural liaison (person performing the link between two sites leveling the cultural differences and language barriers) would bring benefits to the process. The proposed practices and additional roles are suggested based on key findings in the process description and the implemented project documentation. Ideas and approaches are based on the agile principles and the agile manifesto [10] and taking into account the issues of distributed global software development.

9 Future Improvements

As “the new agile process for distributed projects” is a very new software development process, it is hard to collect appropriate amount of statistical information according to proper application of the practices integrated in this process. So, the first thing that can be done to improve it in the near future is to use of this process in running software projects and gathering data about application of every practice. The attention should be focused on gathering the information not only from onsite or offshore, but from both, as this process aims to satisfy all the participants of the development.

As well it is very important to measure the benefits, brought by one or another activity quantitatively and compare the outcome with invested resources. For example, consider the knowledge transfer activity. It is obvious, if the whole offshore development team would be moved onsite to take part in knowledge transfer activity it would be more efficient than in the case when only designers will visit onsite part. But every person transferred from one country to another for one or two weeks requires significant investment of resources. However, future research can aim to find the dependency of knowledge transfer quality from transferred people quantity and find the optimal measure for resource investment in this activity.

As issues of communication create considerable (or even major) amount of problems appearing in global software development, strong impact during future improvement should be done to find out appropriate communicating tools. As well tools for tracking of development process (for example tool, which enables the developer to check the hierarchy) should be found out.

As it was mentioned above authors of “the new agile process for distributed projects”, are satisfied with the way of task division within the team (tasks are allocated by designer or team leader according to his estimations). However, there are other different methods for these purposes (for example Scrum poker). Right now there is not enough statistical information about success of one or another task division method, so in the future appropriate information and statistics could be collected and analyzed. That could result a modification of existing method.

Addressing to the point 6.5 of analysis and discussion part of this report, the issue of excessive or insufficient communication can be identified. As a result the questions such as “How to define appropriate amount of communication between two sites?” and “How to distinguish the questions that should and should not be addressed from offshore part of the team to onsite part of the team?” may arise.

10 References

[1] Maria Paasivaara, Nico Hiort af Ornäs, Peitsa Hynninen, Casper Lassenius, Tuomas Niinimäki, Arttu Piri; "PRACTICAL GUIDE TO MANAGING DISTRIBUTED SOFTWARE DEVELOPMENT PROJECTS"; Helsinki University of Technology Software Business and Engineering Institute Technical Reports C12 Espoo 2010.

[2] Robert Martignoni; "Global sourcing of software development – a review of tools and services"; 2009 Fourth IEEE International Conference on Global Software Engineering.

[3] Ita Richardson, Miriam O'Riordan, Valentine Casey, Bridget Meehan ,Ivan Mistrík, "Knowledge Management in the Global Software Engineering Environment", 2009 Fourth IEEE International Conference on Global Software Engineering.

[4] Juyun Cho, "GLOBALIZATION AND GLOBAL SOFTWARE DEVELOPMENT", Issues in Information Systems, Volume VIII, No. 2, 2007, Page 287-90.

[5] Pär J. Ågerfalk, Brian Fitzgerald, Helena Holmström Olsson and Eoin Ó Conchúir, "Benefits of Global Software Development: The Known and Unknown", Q. Wang, D. Pfahl, and D.M. Raffo (Eds.): ICSP 2008, LNCS 5007, pp. 1 – 9, 2008.

[6] Carmel, E.: Global Software Teams: Collaborating Across Borders and Time Zones, 1st edn. Prentice-Hall, Upper Saddle River (1999).

[7] Erran Carmel, Ritu Agarwal, "Tactical Approaches for Alleviating Distance in Global Software Development", IEEE SOFTWARE March/April 2001, Page 22-29.

[8] James D. Herbsleb and Deependra Moitra, " Global Software Development", IEEE SOFTWARE March/April 2001, Page 16-20.

[9] The Blending of Traditional and Agile Project Management By Kathleen B. Hass, PMP, Project Management Practice Leader, Management Concepts Exclusively for Project Management World Today.

[10] Agile Manifesto, http://www.agilemanifesto.org , last accessed: 2010-03-21.

[11] Ken Schwaber, Mike Beedle, "Agile Software Development with Scrum", Prentice Hall, 2001.

[12] Jay Lundell, Mark Notess "Human factors in software development: models, techniques, and outcomes", Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology, New Orleans, Louisiana, United States, Pages: 145 – 151, year of Publication: 1991

[13] Kniberg, Henrik, "Scrum and XP from the Trenches", 2007.

[14] Yin, R.K. (2003), Case study research – design and methods,Third edn.; Thousand Oaks, California: Sage Publication.

[15] Sounders, M. Lewis P. & Thornhill A. (2000), Research Methods for Business students,Second edn.; Essex; Pearson Education Limited.

[16] Bockmon, D.F., & Rieman, D.J. (1987); Qualitative versus quantitative research; Holistic Practice.

[17] Mack N., Woodsong C., Macqueen K., Guest G., Namey E. (2005), Qualitative Research Methods: A Data Collector's Field Guide (Family Health International, Research Triangle Park, North Carolina, USA.

[18] Greenhalgh T., Taylor R. (2006), How to read a paper: Papers that go beyond numbers (qualitative research) (Unit for Evidence-Based Practice and Policy, Department of Primary Care and Population Sciences, University College London Medical School/Royal Free Hospital School of Medicine, Whittington Hospital, London.

[19] Dale T. Griffee, Research Tips: Interview Data Collection, Journal of Developmental Education, Volume 28, Number 3, spring 2005.

[20] http://www.qualres.org/HomeSemi-3629.html

[21] Cem Kaner, James Bach, and Bret Pettichord 2001. Lessons Learned in Software Testing. John Wiley & Sons, Inc.

[22] Brian White and Geoff Glemm 2000. Software Configuration Management Strategies and Rational ClearCase: A Practical Introduction. Addison-Wesley Longman.

[23] Philippe Kruchten; Architectural Blueprints—The "4+1" View Model of Software Architecture; Rational Software Corp;Paper published in IEEE Software 12 (6) November 1995, pp. 42-50.

[24] Ade Miller, "A Hundred Days of Continuous Integration," agile, pp.289-293, Agile 2008.

[25] http://www.agiledata.org/essays/tdd.html#PartingThoughts

[26] Kent Beck; Test Driven Development: By Example

[27] Meyer, Bertrand: Applying "Design by Contract", in Computer (IEEE), 25, 10, October 1992, pages 40-51

[28] Smith MF Software Prototyping: Adoption, Practice and Management. McGraw-Hill, London (1991).

[29] http://agilesoftwaredevelopment.com/xp/practices

[30] Kent Beck, Addison Wesley; Extreme Programming Explained: Embrace Change; 2000.

[31] http://www.extremeprogramming.org/

[32] http://www.scrum.org

[33] Scrum: Developed and sustained by Ken Schwaber and Jeff Sutherland;scrum.org; February 2010.

[34] Bruce Powel Douglass;ROPES:Rapid Object-Oriented Process for Embedded Systems;I-Logix Inc;adapted from Doing Hard Time:Developing Real-Time Systems using UML, Objects, Frameworks, and Patterns Reading, MA: Addison-Wesley, 1999.

[35] Rational Unified Process:Best Practices for Software Development Teams; Rational Software White Paper TP026B, Rev 11/01.

[36] Eduardo Miranda; Combining Critical Chain Planning and Incremental Development in Software Projects; originally published as a part of 2004 PMI Global Congress Proceedings – Europe.

11 Appendixes

According to the policies of the Northern European Company where this master thesis was performed, authors are not allowed to attach internal documents as appendixes.