Meshing Agile and Documentation-Driven Methods in Practice

Heeager, Lise Tordrup

Ph.D. Thesis

Meshing Agile and Documentation-Driven Methods in Practice

By

Lise Tordrup Heeager

ABSTRACT

The motivation for this thesis originates in software development practice - in the increasing interest in the adoption and diffusion of agile methods in software organizations; even by organizations complying with quality standards, representing the documentation-driven approach to developing software. While agile methods promise to deliver software with flexibility and without excessive cost; documentation-driven software development methods promise predictability and stability. Examples of agile methods are Scrum and eXtreme Programming. An example of a documentation-driven method is the Structured Systems Analysis and Design Method; documentation-driven methods are also associated with the Waterfall model and various quality assurance standards such as the Capability Maturity Model Integration and ISO standards. The agile and the documentation-driven methods appear contradicting at first, but the literature reports on a few scattered examples of software organizations that have succeeded in implementing a software practice meshing agile and documentation-driven methods. However, due to a lack of empirical research documenting this compatibility, it is not well understood how to implement a software development practice that meshes agile and documentation-driven methods.

This thesis addresses the research question: “How can the agile and the documentation-driven methods be meshed in practice?” The research builds on a literature review (providing a theoretical background) and two case studies of the development of safety-critical software. In safety-critical software development, the software practice is by law subject to strict requirements from quality standards, it is therefore discussed whether or not agile methods are suitable in these cases. The first case study presents the challenges of adopting quality assurance in an agile software practice in a small software organization, while the second case study reports on how a software practice meshes the practices and principles of the agile method of Scrum and the requirements and recommendations of the US Food and Drug Administration standard.

Based on the literature review nine elements of agile and documentation-driven methods were identified: 1) management style, 2) customer relations, 3) people-issues, 4) documentation, 5) requirements, 6) development strategy, 7) communication and knowledge sharing, 8) testing and 9) culture. As this thesis is concerned with practice, in the discussion these are referred to as practice areas. The case studies provided findings on how to mesh six of the nine practice areas. Meshing the practice area of customer-relations is difficult. An agile customer relation can hinder the implementation of the documentation-driven customer strategy due to a high level of trust between the parties. Implementing agile customer relations can also be difficult in a documentation-driven project as such a relation is not prioritized by the management. The practice area of documentation is the hardest to mesh; the amount of documentation required by documentation-driven methods is not supported by the agile methods. The practice area of requirements is difficult to mesh. The challenge is related to the documentation of these. The practice area of development strategy can be meshed. A software organization can mesh the agile and the documentation-driven methods by dividing the software practice into short iterations within long-term milestones. The iterations will however be affected by the milestones as the passing of these are dependent on certain activities, for example writing documentation. The practice area of communication and knowledge sharing strategies can be meshed, but will most likely be dominated by the documentation-driven codification strategy. The practice area of testing is difficult to mesh. Both developers and management need to realize the advantages of test-driven software development.

Keywords: Agile Software Development, Documentation-Driven Software Development, Safety-Critical Software Development, Scrum, XP, FDA, CMMI.

RESUMÈ

This is a Danish translation of the abstract.

Motivationen for denne afhandling er funderet i softwareudviklingspraksis - i den stigende interesse i at implementere og udbrede agile metoder i software organisationer, selv i organisationer, der anvender kvalitetsstandarder, som repræsenterer den dokumentationsdrevne måde at udvikle software. Mens agile metoder lover en fleksibel softwareproces, uden overdrevne omkostninger; lover dokumentationsdrevne softwareudviklingsmetoder forudsigelighed og stabilitet. Eksempler på agile metoder er Scrum og eXtreme Programming. Et eksempel på en dokumentationsdrevet metode er Struktureret System Analyse og Design; dokumentationsdrevet softwareudvikling er også forbundet med Vandfaldsmodellen og forskellige standarder for kvalitetssikring såsom the Capability Maturity Model Integration og ISO-standarder. Umiddelbart synes de agile og de dokumentations-drevne metoder modstridende, men litteraturen rapporterer om få spredte eksempler på softwareorganisationer, der i praksis har implementeret en softwareproces, der blander de agile og de dokumentationsdrevne metoder. Men, på grund af få empiriske studier, er denne kompatibilitet ikke dokumenteret og der er stor usikkerhed om, hvordan man kan implementere en software udviklingsproces, der blander de agile og de dokumentationsdrevne metoder.

Denne afhandling adresserer forskningsspørgsmålet: "Hvordan kan de agile og de dokumentationsdrevne metoder blandes i praksis?". Forskningen bygger på et litteraturstudie (der danner det teoretiske fundament) og to casestudier af udviklingen af sikkerhedskritisk software. I udvikling af sikkerhedskritisk software er softwarepraksissen efter lovgivningen underlagt strenge krav fra kvalitetsstandarder. Derfor diskuteres det stadig, om de agile metoder er velegnede i disse tilfælde. Det første casestudie præsenterer udfordringer forbundet med indføringen af kvalitetssikring i en agil softwareudviklingspraksis i en lille softwareorganisation, mens det andet casestudie viser, hvordan en softwareproces anvender en blanding af principper og praksisser fra den agile metode Scrum og kravene og anbefalingerne fra den amerikanske Food and Drug Administration standard.

På baggrund af litteraturstudiet blev 9 elementer af de agile og dokumentationsdrevne metoder identificeret: 1) ledelsesstil, 2) kunderelationer, 3) syn på mennesker, 4) dokumentation, 5) krav, 6) udviklingsstrategi, 7) kommunikations- og vidensdelingsstrategi, 8) test og 9) kultur. Da, denne afhandling ser på praksis, anses disse i diskussionen som værende praksisområder. Casestudierne svarede på, hvordan seks af disse ni praksisområder kan blandes. At blande praksisområdet omkring kunderelationer er vanskeligt. Agile kunderelationer kan på grund af en stor tillid mellem parterne hindre implementeringen af dokumentationsdrevne strategier for håndtering af kunder. Implementeringen af agile kunde relationer i et dokumentationsdrevet projekt kan også være vanskeligt, da sådanne relationer ikke er prioriteret af ledelsen. Praksisområdet omkring dokumentation er det vanskeligste at blande; mængden af dokumentation krævet af de dokumentationsdrevne metoder understøttes ikke af de agile metoder. Praksisområdet omkring krav er vanskeligt at blande. Udfordringerne er relateret til udfordringerne omkring dokumentation. Praksisområdet omkring udviklingsstrategi kan blandes. En softwareorganisation kan blande de agile og de dokumentationsdrevne metoder ved at opdele softwarepraksissen i korte iterationer indenfor langsigtede milepæle. Iterationerne vil dog blive påvirket af milepælene, da bestemte aktiviteter, for eksempel skrivning af dokumentation, skal udføres før milepælen kan passeres. Praksisområdet omkring kommunikations- og vidensdelingsstrategi kan blandes, men vil sandsynligvis være domineret af den dokumentationsdrevne kodificeringsstrategi. Praksisområdet omkring test er vanskeligt at blande. Både udviklere og ledelse skal anerkende fordelene ved testdrevet software udvikling.

Nøgleord: agil softwareudvikling, dokument-drevet softwareudvikling, sikkerhedskritisk softwareudvikling, Scrum, XP, FDA, CMMI

ACKNOWLEDGEMENTS

Writing this thesis has been a large part of my life for the last three years and many people have supported me on the long and somewhat winding road.

First of all I would like to thank Peter Axel Nielsen for his most valuable supervision during these three years and for co-authoring the third paper: “Agile Software Development and its Compatibility with a Document-driven Approach? A Case Study,” and the fifth paper: “Barriers to Adoption of Agile Software Development Practices: A Knowledge Transfer Framework”; I have really valued his feedback and cooperation.

Secondly, I would like to thank Gitte Tjørnehøj for co-authoring the second paper: “Adopting Quality Assurance Technology in Customer-Vendor Relationships: A Case Study of How Interorganizational Relationships Influence the Process”; a process in which she taught me a lot about writing academic papers. I would also like to thank her for supporting and mentoring me all the way through the process, even though she was not my supervisor. She has provided me with valuable support and encouragement.

I would also like to thank my colleagues, especially the other PhD students in the IS group at Cassiopeia (both former and present); it has been a pleasure coming to work every day being able to share both inspiring discussions and fun.

Finally, my husband Jacob, my family and friends have also been a valuable support during this PhD journey. Thank You!

TABLE OF CONTENTS

1. INTRODUCTION 1

1.1 MOTIVATION 1
1.2 MESHING AGILE AND DOCUMENTATION-DRIVEN METHODS 2
1.3 STRUCTURE OF THE THESIS 2

2. THEORETICAL BACKGROUND 3

2.1 SOFTWARE DEVELOPMENT METHODS AND PRACTICE 3
2.2 AGILE SOFTWARE DEVELOPMENT 3
2.3 DOCUMENTATION-DRIVEN SOFTWARE DEVELOPMENT 8
2.4 DIFFERENCES AND SIMILARITIES OF THE AGILE AND THE DOCUMENTATION-DRIVEN METHODS 12
2.5 MESHING THE AGILE AND THE DOCUMENTATION-DRIVEN METHODS 15
2.6 RESEARCH QUESTION 16

3. THEORETICAL FRAMEWORKS 17

3.1 THE AGILE AND THE DOCUMENTATION-DRIVEN HOME GROUNDS 17
3.2 INTERORGANISATIONAL RELATIONSHIPS: ATTRIBUTES AND PROCESSES 19
3.3 KNOWLEDGE TRANSFER 21

4. RESEARCH APPROACH 23

4.1 RESEARCH DESIGN 23
4.2 LITERATURE REVIEW 25
4.3 CASE STUDIES 26

5. RESEARCH SUMMARY 33

5.1 OVERVIEW OF THE PAPERS 33
5.2 THE AGILE AND THE DISCIPLINED SOFTWARE APPROACHES: COMBINABLE OR JUST COMPATIBLE? 34
5.3 ADOPTING QUALITY ASSURANCE TECHNOLOGY IN CUSTOMER-VENDOR RELATIONSHIPS: A CASE STUDY OF HOW ORGANIZATIONAL RELATIONSHIPS INFLUENCE THE PROCESS 34
5.4 AGILE SOFTWARE DEVELOPMENT AND ITS COMPATIBILITY WITH A DOCUMENT-DRIVEN APPROACH? A CASE STUDY 35
5.5 INTRODUCING AGILE PRACTICES IN A DOCUMENTATION-DRIVEN SOFTWARE DEVELOPMENT PROCESS: A CASE STUDY 36
5.6 BARRIERS TO ADOPTION OF AGILE SOFTWARE DEVELOPMENT PRACTICES: A KNOWLEDGE TRANSFER FRAMEWORK 37

6. DISCUSSION 39

6.1 MESHING 39
6.2 THE OVERALL MESH IN THE CASE STUDIES 40
6.3 THE MESH OF EACH PRACTICE AREA IN THE CASE STUDIES 42
6.4 LIMITATIONS 53
6.5 FUTURE RESEARCH 54

7. CONCLUSION 57

REFERENCES 59

APPENDIX: THE PAPERS 67

1. INTRODUCTION

This chapter describes the motivation for this thesis, the area of concern and the structure of the thesis. The motivation originates in software development practice - in the increasing interest in the adoption and diffusion of agile methods in software organizations; even by organizations complying with quality standards, representing the documentation-driven way of developing software. The agile and the documentation-driven methods seem contradicting at first, but the literature reports on a few scattered examples of software organizations that have succeeded in implementing a meshed software practice.

1.1 Motivation

Within software development, much focus has been and still is on the methods for developing software. From the beginning, software developers have struggled to find the best way to develop software whilst avoiding serious flaws and errors. Over time, software projects have grown in both complexity and time frame, meaning that software development has become even more difficult. Problems regarding the cost, timeliness, and quality of software products still exist (Iivari, Iivari 2010). This has led to considerable attention on the development of software, and several methods have been introduced over the past 30 years (Hirschheim et al. 1995). Analysing the history of software development methods, Avison and Fitzgerald (2003) have identified four eras:

1) the 'Pre-Methodology Era' (1960's and 1970's), in which the early computer applications were developed without explicit and formalised development methods,
2) the 'Early Methodology Era' (late 1970's, early 1980's), which was characterised by the systems development life cycle also known as the Waterfall model,
3) the 'Methodology Era' in which a number of new methods emerged as a response to the limitations and short-comings of the Waterfall model and
4) the 'Post-Methodology Era' (starting late 1990's) in which some organizations continued to try to implement even newer methods, whilst many other organizations had given up on the methods, mainly because they had problems implementing them.

Avison and Fitzgerald end up by concluding that, currently, methods may be used more than ever. The matter of choosing the appropriate software development method is something which continuously occupies many IT practitioners (Fitzgerald et al. 2002). Systems development methods are therefore still significant in research and practice (Iivari, Iivari 2010).

Documentation-driven software development is often symbolised by the Waterfall model, consisting of a linear development strategy of analysing, designing, implementing and testing. Another example is Structured Systems Analysis and Design Method (Hares 1994). Documentation-driven development is also associated with quality standards such as CMMI (Chrissis et al. 2003) and the ISO standards (Hoyle 2006), developed to guide the software development practice. The failures of documentation-driven methods resulted in the development of agile methods, and in 2001 the agile manifesto was written. Neither the agile nor the documentation-driven methods provide the ultimate method; both have shortcomings and pitfalls (Boehm, Turner 2003a). Agile software development methods promise a way to deliver software without excessive cost (Boehm, Turner 2004). Examples of agile methods are Scrum (Schwaber, Beedle 2001) and eXtreme Programming (XP) (Beck, Andres 2004). The agile methods embrace change (Beck, Andres 2004) and work well in areas of high uncertainty, but they lack the ability to handle high complexity (Dahlbom, Mathiassen 1993). In contrast, the documentation-driven software development methods are well-defined (Nawrocki et al. 2002) and work well for projects with high complexity, but have difficulties dealing with high uncertainty and change (Dahlbom, Mathiassen 1993). Consequently, which method should we choose for a project with high complexity and high uncertainty? Interest in the possibility of meshing agile and documentation-driven methods has arisen; either by a wish to implement agile elements in a documentation-driven practice in order to for example increase flexibility or by a wish to implement documentation-driven elements in an agile practice in order to for example comply with a quality standard..

1.2 Meshing Agile and Documentation-Driven Methods

As the agile and the documentation-driven methods are two different ways of developing software (Dahlberg et al. 2006), they appear contradicting (Turner 2002; Turner, Jain 2002). One of the differences between the agile and the documentation-driven methods suggested by the literature is the amount of documentation. Agile methods do not support the degree of documentation required by documentation-driven methods (Boehm, Turner 2005; Paetsch et al. 2003). Both agile and documentation-driven methods are, however, flexible and adaptable, research is therefore looking at the possibility of meshing agile and documentation-driven methods (Rönkkö et al. 2008; Zanatta, Vilain 2006). Several researchers agree that in practice it is possible to create a software practice that meshes agile and documentation-driven principles and practices (e.g. Baker 2005; Boehm, Turner 2003a; Nawrocki et al. 2006; Stephenson et al. 2006), but the majority also agree that either the agile method, the documentation-driven method or both methods need an extension in order to make them compatible. Some researchers state that the agile and the documentation-driven methods have much in common, and that such a meshed practice is preferable as it will be able to take advantage of the strengths and compensate for the weaknesses of the methods (e.g. Boehm, Turner 2003a; Galal-Edeen et al. 2007; Navarrete et al. 2006). Documentation-driven methods support the agile methods by providing a disciplined framework (Elshafey, Galal-Edeen 2008) and by providing structure to ensure the agile principles and practices are followed (McMichael, Lombardi 2007). However, as the agile and the documentation-driven methods are grounded in opposing concepts, meshing them is not straight forward (Opelt, Beeson 2008; Reifer 2003; Vinekar et al. 2006), and because of the scarcity of empirical studies on the subject it is not well understood how to achieve this in practice. The purpose of this study is therefore to gather empirical evidence on the possibility of meshing agile and documentation-driven methods in software development practice.

1.3 Structure of the Thesis

In the next chapter the theoretical background for the thesis will be presented, it includes a review of the literature. It provides a section on software development methods and practice, a presentation of agile software development, a presentation of documentation-driven software development, an identification of the differences and similarities of the agile and the documentation-driven methods and reviews previous literature on the meshing of these methods. Based on this theoretical background and the identified knowledge gaps, the research question is formulated and presented. In the third chapter the theoretical frameworks are presented, the frameworks are developed in order to support each other to uncover the knowledge that answers the research question. The research approach is presented in chapter four. The thesis consists of five individually published papers and this summary; in section five: ‘Research Summary’ the five papers and their coherence are presented. Finally, the findings are discussed and conclusions provided in chapter six and seven. The five papers are included in the appendix.

2. THEORETICAL BACKGROUND

This chapter presents the theoretical background. First, a section on software development methods and practice is presented; followed by a description of agile software development and of documentation-driven software development. After which, the differences and similarities between the agile and the documentation-driven methods and the existing literature on meshing agile and documentation-driven software methods are described. Finally, the research question is presented.

2.1 Software Development Methods and Practice

The terms methods and practice are closely connected and often used interchangeably in the literature, thus definitions of these are needed. Many definitions of methods exist in the literature (e.g. Avison, Fitzgerald 2006; Hirschheim et al. 1995). This thesis draws on the definition by Nielsen (1991, p. 36); methods consist of:

- Domain of use and modelling languages (Product-oriented),
- Frame of action and related techniques (Process-oriented),
- Perspective.

This is an extension of the definition provided by Mathiassen (Mathiassen 1982). The domain of use is the range of situations in which the method is useful; the modelling languages are a set of concepts and rules for the use. The frame of action is activities recommended when using the method. The perspective is the set of views and beliefs that makes the method meaningful.

Practice is: "what actually happens in development, rather than what ought to or should happen according to the method" (Omland 2009, p. 4); thus the actual software development situation. Methods influence practice as they provide guidance for the organization (Omland 2009). Several empirical studies do, however, show that in practice developers rarely adopt methods in the entirety (Fitzgerald 1997; Madsen et al. 2006; Mathiassen, Purao 2002). Software organizations adapt the methods and only apply elements of the method (Madsen et al. 2006). Developers are modifying and omitting aspects of the methods, tailoring the methods to fit precisely to the development environment (Fitzgerald 1997). Thus, there is an important difference between methods and how they are applied in practice (Mathiassen, Purao 2002). Fitzgerald (1997) introduces the notion methods-in-action in order to describe the relationship between methods and their use in practice. Madsen and Kautz (2006) use the term emergent methods; they define these as: "the actual unfolding development process and the activities, and applied method elements that comprise this process" (p. 226). They also highlight that a formalized method should by the organization be regarded as a guide rather than a prescriptive basis for planning and action (Madsen et al. 2006). This shows that it, in this thesis, is relevant to look at the methods in relation to practice.

2.2 Agile Software Development

Agile software development is becoming increasingly popular (Misra et al. 2010), and many organizations are trying to adopt these to take advantage of the numerous benefits they offer (Sidky et al. 2007). The agile software methods are derived from the weaknesses and failures of the documentation-driven methods (Taylor et al. 2006). In 2001, the agile manifesto was created; it constitutes four values and a set of practices for software development. The four values are: 1) individuals and interactions over processes and tools 2) working software over comprehensive documentation 3) customer collaboration over contract negotiation 4) responding to change over following a plan (Beck et al. 2001).

Many terms have been used for the agile methods, including Internet-speed, lightweight, and lean methods. Agile is the term preferred by the Agile Alliance (Agile Alliance 2011), a group of software professionals promoting the concepts of agile software development (Paulk 2002). Since the term agility has been identified in a variety of ways in the literature (Sarker et al. 2009), a better understanding of what constitutes agility is required (Abrahamsson et al. 2009). Mathiassen and Pries-Heje (2006, p. 116) associate business agility with responsiveness to customer requests, market dynamics, and emerging technology options (quickness) and responsiveness to changing demands (adaptability). They define an agile organization as an organization, that is "nimble, capable of responding quickly and gracefully to both expected and unexpected events in their environment." In the context of information systems development, researchers agree that the core elements of the concept of agility are 1) a capability for sensing, 2) a capability to respond to change and 3) the use of lightweight methods and practices (Sarker et al. 2009). Sambamurthy et al. (2003, p. 245) define agility as "the ability to detect opportunities for innovation and seize those competitive market opportunities by assembling requisite assets, knowledge, and relationships with speed and surprise". Whereas, Lee et al. (2006b, p. 50) define information systems agility as "the ability of information systems development and deployment methods to swiftly adapt to the changing business requirements". Qumer and Henderson-Sellers (2008, p. 281) have defined an agile method as "people-focused, communications-oriented, flexible, speedy, lean, responsive and learning". It has to be ready to adapt to any change at any time, encourage rapid and iterative development, focus on shortening the timeframe and cost, and focus on improvement both during and after product development. In these definitions, agility in software development is associated with adaptability (responsiveness to change) and the employment of a method that is capable of supporting this adaptability. This thesis adopts the definition of agility provided by Conboy (2009); it supports the earlier definitions of agility and is established on a strong methodological base. It takes its starting point in the concepts of flexibility and leanness:

> "the continual readiness of an ISD method to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while contributing to perceived customer value (economy, quality and simplicity), through its collective components and relationships with its environment." (Conboy 2009, p. 340)

Agile Methods

The Dynamic Systems Development Method (DSDM) (Stapleton 1999) has been identified as the first agile method (Dybå, Dingsøyr 2008); it was followed by eXtreme Programming (XP) (Beck, Andres 2004). Through the years, other agile methods have appeared: the Crystal family of methods (Cockburn 2006), Evolutionary Project Management (EVO) (Gilb 2006), Lean Development (Poppendieck, Poppendieck 2003; Poppendieck 2007) and Scrum (Schwaber, Beedle 2001). See table 1 for an overview of the methods. Today, Scrum and XP are the two most well-known and popular agile methods. While Scrum provides guidance for the efficient management of projects (Jakobsen, Johnson 2008), XP offers a range of practices and principles to apply (Beck, Andres 2004) in order to create flexibility and adaptability.

Agile method	Description
Dynamic Systems Development Method (DSDM)	DSDM is about people; it is about understanding the needs of a business delivering software solutions as quickly as possible.
eXtreme Programming (XP)	XP is an iterative and incremental method, that emphasises customer contact and a flexible response to change.
Crystal	The Crystal family of agile methods provides different methods corresponding to the size and criticality of the project.
Evolutionary Project Management (EVO)	EVO is not just a method for software but also for larger systems engineering contexts; it is iterative, focusing on the customers' highest prioritised requirements.
Lean Development	Lean Development provides the theory behind agile software development; it gives organizations a set of principles to minimise waste and cost.
Scrum	Scrum is an iterative and incremental method, that emphasises project management values and practices.

Table 1. An overview of agile methods

Each of the agile methods has specific features, but they do share some common characteristics (Beznosov, Kruchten 2004): Through an iterative, test-driven software development strategy with frequent customer feedback, the agile methods seek high software quality. The short iterations, multi-disciplinary teams, knowledge sharing and continuous integration allow better control over the project and increase visibility (Mahanti 2004). To enhance knowledge sharing, they advocate an informative workspace that includes information radiators (Cockburn 2006). This could for example include task boards and burn down charts placed where passersby can see them. Agile methods rely on the competencies of the software developers (Aaen 2003). Refactoring (redesigning and rewriting software) is also advocated by the agile methods, as it serves to correct poorly written and redundant code (Vogel 2006). It is however discussed whether agile software development provides the quality assurance provided by the documentation-driven software development methods.

eXtreme Programming

The goal of XP is that software can be developed with lower cost, with fewer defects, with higher productivity and with a much higher return on investment (Beck, Andres 2004). XP consists of values, principles and practices. The values are: communication, simplicity, feedback, courage and respect. The values and practices are connected through principles such as improvement and reflection. Table 2 summarises the most notable practices of XP. The software is developed in iterations (cycles), two different cycles are suggested, weekly and quarterly. The purpose of the planning game is to gather, prioritise and estimate the tasks at hand; this is done by the customer or at least in very close cooperation with the customer. The requirements are written by the customer in a natural language, these are referred to as user stories. The goal of test-driven development is to reduce costs and defects, the recommendations are that the test cases are written before the code is implemented, the tests are run at every build of the software, and that the tests are run automatically. The new code is continuously integrated into the code base. Refactoring is a frequent redesign of the system, which makes sure that the code is constantly improved and the principle of simplicity is maintained. In order to conduct frequent reviews of the code, pair programming is recommended. The XP team consists of a variety of people working closely together, for example: developers, testers, interaction designers, architects and users. The code is shared by all team members and can be changed by anyone at any time. The customer is important in XP that recommends real customer involvement and if possible includes a full-time, on-site customer.

Practice	Description
Weekly cycle	Plan work a week at a time. Have a meeting at the beginning of every week.
Quarterly cycle	Plan work a quarter at a time.
Planning game	The planning game is used at the beginning of each release and iteration. Customers decide on the scope and timing of releases based on estimates provided by programmers. Developers only implement the functionality demanded by the user stories in each iteration.
Test-driven development	XP advocates that tests are done early, often and automated.
Continuous integration	The new code is integrated into the current system after no more than a few hours.
Refactoring	The design of the system evolves through transformations of the existing design that keeps all the tests running.
Pair programming	All production code is written by two people at one machine.
Whole team	A variety of people working together.
Shared code	Anyone on the team can improve any part of the system at any time.
Real customer involvement	Whole team implies customer involvement; ideally the customer is present on-site and works with the team full-time.

Table 2. eXtreme programming practices (Beck, Andres 2004)

Scrum

Scrum is an empirical approach that reintroduces flexibility, adaptability, and productivity into systems development (Schwaber, Beedle 2001). Scrum is a management and control method that cuts through complexity to focus on building software that meets business needs. It includes the roles of the Scrum master, the product owner and the team. It furthermore consists of the practices: sprints, daily Scrum meetings, sprint planning meetings, sprint reviews and the backlogs. Table 3 summarises the practices and roles of Scrum. The Scrum master is responsible for managing the software development practice, making sure that the Scrum values, practices and rules are enforced. The product owner is officially responsible for the project, and ideally this role is filled by the customer. The Scrum team is the developers and testers and other people working on the software. A sprint is a fixed period of time for which the team works; Scrum recommends 30 day sprints in which there is no interference. Every day a Scrum meeting is held; it is a status meeting of a maximum of 15 minutes. At these meetings the managers can get a sense of what and how the team is doing. At the sprint planning meeting the functionality of the next sprint is determined. The sprint review is a meeting in which the Scrum team presents what has been built during the sprint. Scrum operates with three backlogs, the product backlog (all product requirements), the release backlog (the tasks chosen for the next release) and the sprint backlog (containing the tasks for the current sprint).

Role / practice	**Description**
Scrum master	The Scrum master is responsible for the success of Scrum.
Product owner	The oroduct owner is the customer of the product; he alone controls the product backlog.
Scrum team	Self-organising project teams.
Sprints	The team works for a fixed period of time called a sprint.
Daily Scrum meeting	Each Scrum team meets daily for a 15-minutes status meeting, these meetings are often done standing up, at it helps keep them short.
Sprint planning meeting/day	Customers, users, management, the Product owner and the Scrum team determine the next sprint goal and functionality at the sprint planning meeting. The team then devises the individual tasks that must be performed to build the product increment.
Sprint review meeting	During this meeting, the team presents to the management, customers, users and the Product owner the product increment that has been built during the sprint.
Product backlog	The product backlog is a prioritised list of all product requirements.
Release backlog	Release backlog is that subset of the product backlog that is selected for a release.
Sprint backlog	Sprint backlog consists of the tasks that the Scrum team has devised for a sprint.
Scrum boards	Task boards displaying the progress of the sprint is placed in the office where the developers are able to see them easily.

Table 3. Roles and practices of Scrum (Kniberg 2007; Schwaber, Beedle 2001)

Crystal

The Crystal family of agile methods was developed by Cockburn (2005). Recognising that different projects need to be run differently, each method is adjusted to fit the particular setting. As with geological crystals, each Crystal method has a different colour and hardness, corresponding to the size and criticality of the project (Cockburn 2006). Cockburn has created a scale to illustrate the matching of method configurations to these factors (see figure 1). For example, Crystal Clear is useful for small software teams and projects of low to medium criticality (C6, D6 and E6), while Crystal Orange is useful for small to medium sized software teams and for low to medium criticality (C20, D20, E20). All of the Crystal methods are iterative, and people and communication-centric. Cockburn (2005) presents six properties of the Crystal methods:

1) frequent delivery: the single most important property of any project, large or small, agile or not, is of delivering running, tested code to real users every few months,
2) reflective improvement: the ability to identify what is and is not working, discuss what might work better, and make those changes in the next iteration,
3) osmotic communication: information flows are picked up by the members of the team as though by osmosis,
4) personal safety: being able to speak without fear of reprisal,
5) focus: knowing what to work on, and having the time and peace of mind to work on it, and
6) easy access to expert users: access to an expert user is immensely valuable.

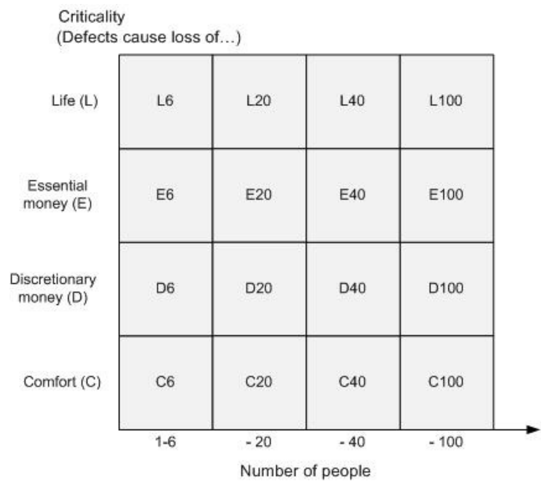


Figure 1. The Crystal methods

2.3 Documentation-Driven Software Development

The objectives of documentation-driven methods are to create predictability, repeatability and optimisation (Boehm 2002). The methods aim at achieving this by being highly plan-oriented (Dahlbom, Mathiassen 1993); by focusing on long term plans controlled by long-term milestones (Boehm 2002). They rely heavily on a documentation strategy; in addition to the end product these methods produce documentation of different aspects of the software development and the product (Nerur et al. 2005). The requirements, the architecture and design and the code are for example documented. The documents are both used as a knowledge sharing strategy and as proof that all procedures have been followed, for example the code review. The project is controlled through a command-and-control style of management, that includes a clear separation of roles (Moe et al. 2010). The customer is included at the beginning of the project, where the requirements are specified and the contract is negotiated (Nerur et al. 2005). The strengths of these methods are the comparability and repeatability they create; since the development practice is defined and specific work products are formatted, it is easy to find the information desired and the estimation of the work becomes more standardised. Applying these methods too rigidly can, on the other hand, result in a mechanical, checklist mentality and impede innovation; the plan may become the focus instead of the product and there is a risk that the developers become documentation generators instead of software developers (Boehm, Turner 2003d). Due to standardisation, documentation-driven methods work well for stable projects, but they lack the ability to handle change (Dahlbom, Mathiassen 1993).

In the literature, researchers use several different terms for documentation-driven methods and models. The majority of researchers refer to these methods as plan-driven methods (e.g. Abrahamsson et al. 2009; Boehm, Turner 2003d; Cohn, Ford 2003); a reference to the heavy reliance on plans. However, agile methods also include a lot of planning; XP for example includes planning activities each day and involves much more planning than many organizations complying with a quality standard do (DeMarco, Boehm 2002). In contrast to the term plan-driven, the agile methods can be referred to as planning-driven (Boehm, Turner 2003d). While the quality standards rely on a master plan, the agile methods are driven and controlled by the continuous planning activities. The term traditional is also popular when referring to these methods (e.g. Galal-Edeen et al. 2007; Misra et al. 2010; Moe et al. 2010). However, using this term indicates that these methods are the ones most used, and that they were the original methods. Disciplined is a third term used: several researchers, however, underline the need for both agility and discipline in a software development practice (e.g. Boehm, Turner 2004; DeMarco, Boehm 2002; Elshafey, Galal-Edeen 2008; Galal-Edeen et al. 2007). Agility and discipline are not necessarily opposites, agile methods are highly disciplined; XP for example provides a clear overview of the activities that are expected (Beck, Boehm 2003). Document-driven is also a highly used term (Boström et al. 2006); that indicates a heavy reliance on documents. But adhering to these methods is not just a matter of producing documents (Wright 2003). Instead, this thesis suggests the use of the term documentation-driven which indicates that the methods rely on documentation in various ways, for example for the sharing of knowledge and for proving the quality of the product. Furthermore, the amount and reliance on documentation seems to be one of the greatest differences between agile methods and the software practice suggested by the quality standards. Using the term documentation-driven therefore underlines and acknowledges the differences between the agile and the documentation-driven methods. It is, however, accepted that agile methods also can contain a lighter amount of documentation, but they are not driven and dependent on documentation. The conceptualisation used in this thesis is therefore documentation-driven.

The literature does not provide sound definitions of documentation-driven methods, as it does with agility. Boehm and Turner (2003d) have however provided a definition of these methods, in their book referred to as plan-driven methods. The definition will in this thesis be used as a definition of documentation-driven methods:

> "Plan-driven methods are characterised by a systematic engineering approach to software that carefully adheres to specific processes in moving software through a series of representations from requirements to finished code. There is a concern for completeness of documentation at every step of the way so that thorough verification of each representation may be accomplished after the fact." (Boehm, Turner 2003d, p. 10-11)

Documentation-Driven Methods

Table 4 gives an overview of documentation-driven methods and models. Documentation-driven development is often associated with the Waterfall model, also known as the systems development life cycle consisting of sequential development stages; one phase has to be implemented before the next phase can begin. Each phase is ended with the approval of the documentation. The traditional Waterfall model consists of five phases: 1) the requirements phase, 2) the design phase, 3) the implementation phase, 4) the test phase and 5) the support phase (Stober, Hansmann 2010). The V-model is a variation of the Waterfall model, it has the same sequential structure, but it links the development phases to the software tests. The left side of the V-model contains the development phases, while the right side contains the corresponding tests (Hilburn, Townhidnejad 2000). The methods Structured analysis (DeMarco 1979) and Modern Structured Analysis (Yourdon 1989) are highly connected; both methods provide structured techniques to the analysis phase. Structured Systems Analysis and Design Method (SSADM) is another example of a documentation-driven method, it is a stage model including the stages of feasibility study, requirements analysis, requirements specification, logical system specification, physical design and the overall stage: plan, monitor and control, i.e. each stage has its own plans, timescales, controls and monitoring procedures (Hares 1994). Systems Analysis and Design (SAD) is a structured approach to systems development, based on the systems development life cycle, it includes the four phases of planning, analysis, design and implementation (Dennis, Wixom 2000). The Rational Unified Process (RUP) is an iterative method; it is adaptable with respect to the amount of documentation produced. But in the early iterations, a main theme is risk-driven development, in these iterations RUP focuses on several work products, such as the software architecture document and the risk list (Kruchten 2004). RUP can therefore, and in many cases is, used as and can be defined as a documentation-driven method.

Documentation-driven method/model	Description
The Waterfall model	The Waterfall model consists of sequential development stages.
The V-model	The V-model has, as the Waterfall model, sequential stages, but these are linked to the different software tests.
Structured analysis and Modern Structured Analysis	Both Structured analysis and Modern Structured Analysis provide structured techniques to the analysis phase
Structured Systems Analysis and Design (SSADM)	SSADM is a stage model, in which each stage has its own plans, timescales, controls and monitoring procedures.
Systems Analysis and Design (SAD)	SAD is a structured approach that includes four phases: planning, analysis, design and implementation
Unified Process (RUP)	RUP is an iterative method, that is adaptable on the amount of documentation. It has a strong focus on risk and can therefore, often, be defined as documentation-driven.

Table 4. Some examples of documentation-driven methods and models

Documentation-driven methods are also associated with different quality and process standards. Just like the documentation-driven methods, these standards contain years of knowledge and experience of systems development placed into a structure, the standards are used in an attempt to ensure the quality of the product. Quality assurance in software development adds activities to the system development practice. The purpose of these activities is to continuously evaluate and correct the software in order to improve the quality assurance (Bang et al. 2002). The documentation is mainly done in order to ensure and prove the quality of the product. It presupposes a structured and systematic effort and approach to heighten the quality of the products and the practice. The continuous measurement and evaluation of the practice is carried out through reviews, tests, process analysis and with the use of metrics.

Table 5 gives an overview of relevant quality standards and process standards. The CMMI (Chrissis et al. 2003) and the ISO9000 family of standards (Hoyle 2006) are well-known examples of standards. CMMI provides insights into what activities are needed to maintain a mature organization capable of predicting and improving the performance of the projects (Jakobsen, Johnson 2008). Organizations rely on documentation-driven methods such as quality assurance for different reasons. Large organizations, for example, need some of the structure they provide. Other organisations rely on these as they are developing software for a particular market, e.g. software development for the pharmaceutical market, which may include safety-critical software. The standards for safety-critical software differ from the standards such as CMMI and ISO, as they have more stringent requirements with regards to the final quality of the software, and therefore have more stringent requirements for the software development practice (Wright 2003). Examples of such standards are the US Food and Drug Administration (FDA) standard and the Good Automated Manufacturing Practice (GAMP) standard.

Quality/process standard	Description
ISO9000	The ISO9000 is a collection of five standards, whereas ISO9000 contains requirements for software development. It contains key operating procedures; such as the development and approval of a quality manual.
FDA	The FDA standard is used for the approval of medical devices for the US market. Approval requires that quality assurance techniques, such as identification of hazards, have been followed.
GAMP	The GAMP standard is used for medical devices for the European market. It also includes several quality assurance techniques, such as strict reviews.
CMMI	The CMMI model is a maturity model that consists of process areas, best practices and generic goals.

Table 5. Some examples of quality and process standards

This thesis focuses on safety-critical software development which by law has to comply with one or more quality standards or process standards. In the cases of safety-critical software development, the documentation-driven methods are represented in the requirements of the standards. The standards therefore become essential. Below the ISO standards, FDA standard, GAMP standard and CMMI model are described in more detail.

The ISO Standards

The ISO 9000 series is an example of well-known and highly used standards. It is a collection of standards for quality assurance and consists of five standards: ISO 9000, ISO 9001, ISO 9002, ISO 9003 and ISO 9004. ISO 9000 is only used for choosing the appropriate standard. ISO 9001 is the only standard containing requirements for the development practice; organisations developing software are therefore to comply with this standard. To comply with an ISO standard it is essential to develop, implement, and maintain a quality manual and key standard operating procedures (definition of key activities and an identification of who is responsible for their implementation). An organisation cannot comply with the requirements of the ISO standard if there is no approved quality manual. The quality policy, goals, and objectives defined in the quality manual must be verifiable. The procedures have to be established and documented by the organisation prior to the certification process (Haider 2001).

The Food and Drug Administration Standard

A medical device for the US market has to be approved by the FDA standard (U.S. Department of Health and Human Services 2010) and for medical devices with software, as part of this approval the software development practice has to be compliant with their process standards. To obtain approval one needs to show that quality assurance techniques have been applied (Lee et al. 2006a). The development therefore includes: 1) identification of the system hazards and a definition of safety requirements, 2) a determination of how the various components in the system can contribute to these hazards, 3) defining derived safety requirements for the components (repeated recursively over the system hierarchy as needed), and then 4) the development of the components to meet these safety requirements (Heimdahl 2007).

The Good Automated Manufacturing Practice Standard

The GAMP (Good Automated Manufacturing Practice) standard is published by the International Society for Pharmaceutical Engineering (ISPE). It is a quality assurance standard for safety-critical, medical devices with software. GAMP is part of the GxP family of standards (Good Practice quality guidelines and regulations) used by the pharmaceutical and food industries, and includes Good Manufacturing Practice (GMP), Good Laboratory Practice (GLP), Good Documentation Practice (GDP) and Good Clinical Practice (GCP) (The International Society for Pharmaceutical Engineering 2010).

The Capability Maturity Model Integration

The Capability Maturity Model (CMM) developed by the Software Engineering Institute (SEI) was replaced in 2002 with the first version of the Capability Maturity Model Integration (CMMI), that combines three previous models into one: the software CMM (SW-CMM), the integrated product development CMM (IPD-CMM), and the software acquisition CMM (SA-CMM). There are therefore a few differences between the software CMM and CMMI; the CMMI contains additional process areas, new best practices and new generic goals. Over the past years, the CMMI have been broadly used for assessing organisational maturity and process capability throughout the world. The CMMI is a model for measuring and improving organisational maturity and capability; its purpose is to guide a software organisation to improve the quality and reliability of its development processes and software. CMMI consists of five maturity levels: initial, managed, defined, quantitatively managed and optimising; each of these levels contains a number of process areas. The process areas describe the goals and activities that make up the process requirements. The CMMI also provides a set of generic practices supporting the improvement of the processes established under the process areas. The model also includes generic goals; goals that appear in multiple process areas. The activities that are expected to result in the achievement of specific goals are called specific practices, and generic practices that appear in multiple process areas are considered important for achieving associated generic goals (Chrissis et al. 2003).

2.4 Differences and Similarities of the Agile and the Documentation-Driven Methods

In order to compare and be able to determine how the agile and the documentation-driven methods can be meshed, it is important to have a detailed understanding of both the differences and similarities of these methods. The agile manifesto is developed as a contrast to the principles of documentation-driven methods, but as the agile methods value most the items on the left side (individuals and interactions, working software, customer collaboration and responding to change), they do also value the items on the right, which are primarily associated with documentation-driven methods (processes and tools, documentation, contract negotiation and following a plan). This indicates that the agile and the documentation-driven methods have several differences, but also some similarities. This section highlights the differences and similarities identified in the literature. Through the review of the literature, nine elements of the methods in which the agile and the documentation-driven methods differ have been identified; these are summarised in table 6 and described in more detail below.

Method element	**Agile software development**	**Documentation-driven development**
Management strategy	Agile methods are based on self-managing teams.	Documentation-driven methods are based on control by management.
Customer relations	Agile methods involve the customer throughout the whole development.	In documentation-driven methods the customer is mainly involved during the early phase of the project.
People-issues	Agile methods focus on the social aspects of software development and people-issues are at the heart of the agile movement.	As documentation-driven methods rely on documentation and standardisation, less experienced developers can contribute.
Documentation	One of the agile principles is: "working software over comprehensive documentation".	Documentation-driven methods use the documents as a proof of quality.
Requirements	Agile methods rely on user stories written by the customer in a plain business-like language as the requirements.	Documentation-driven methods rely on requirements to be defined in the early stages and specified in documents.
Development Strategy	Agile methods propose an iterative development strategy.	Documentation-driven methods propose a sequential development strategy.
Communications and knowledge sharing	Agile methods focus on person-to-person communication (personalization).	Documentation-driven methods focus on documented knowledge (codification).
Testing	Agile methods advocate test-driven software development.	Documentation-driven methods advocate late testing and rely on test plans.
Culture	The agile culture is social and team-orientated.	The documentation-driven culture is plan-orientated.

Table 6. Summary of the differences between the agile and the documentation-driven methods

The agile and the documentation-driven software development methods are based on different project management strategies (Dybå, Dingsøyr 2008). Adopting agile methods in a documentation-driven organisation requires a change from command and control management to leadership and collaboration (Misra et al. 2010; Nerur et al. 2005), which requires a reorientation not only for the developers but also from management (Moe et al. 2010).

Another difference between the agile and the documentation-driven methods is customer relations. Both the documentation-driven methods and the agile methods highly value the satisfaction of the customer. However, in documentation-driven methods the customer is mainly involved during the early phases of the project, while agile methods involve the customer throughout the whole development (Paetsch et al. 2003). The customer involvement in agile methods is therefore much greater (Huo et al. 2004). Establishing and maintaining close and effective customer collaboration has proven difficult and is therefore a significant barrier to the implementation of an agile method (Nawrocki et al. 2002; Paulk 2002).

Software engineering is a knowledge intensive process (Chau et al. 2003), which makes the people associated the most critical success factor (Boehm, Turner 2003b). The agile methods focus on the social aspects of software development (McAvoy, Butler 2009) and people-issues are at the heart of the agile movement (Galal-Edeen et al. 2007). This heavy reliance on human factors poses new challenges when adopting agile practices in a documentation-driven organisation (Esfahani et al. 2010). It is a significant departure from the focus on plans suggested by the documentation-driven methods (Vinekar et al. 2006). A focus on people is therefore necessary (Boehm, Turner 2003a); attention needs to be given to people-related factors such as staffing and communications (Boehm, Turner 2003b). Amicability, talent, skill and communication are critical people factors for agile methods, as proposed by Cockburn and Highsmith (2002). Most research suggests that agile principles needs high-capability people (Nerur et al. 2005), but some cases show that agile projects have succeeded with a mix of inexperienced and very experienced people, and so have documentation-driven projects (Boehm 2002). Besides hiring good people, Boehm and Turner (2005) also advocate that all stakeholders are educated, and that the results of the development practice are rewarded. Education is vital as both the developers and the management need to become familiar with the agile principles and practices before attempting to adopt its practices. Such training is used to eliminate misconceptions and tackle the fear of change. Furthermore, the upper and middle management should be informed of the benefits of agile methods (Mahanti 2004). Cohn and Ford (2003) suggest that adoption of agile practices requires an appropriate combination of scepticism, enthusiasm and cautious optimism from the developers. If people doubt that they will be able to acquire the required agile skills, a fear of change may arise. A fear of change can lead to a protest against the agile practices and an undermining of related efforts (Mahanti 2004). Self-managing teams are essential in agile methods, but can be hard to implement. It is not enough to put people together and label them self-managing, people need training in how to coordinate and work effectively as an agile team (Moe et al. 2010).

The amount of documentation is one of the most significant differences between the agile and the documentation-driven methods; while the agile methods tend not to document enough, the documentation-driven methods tend to be over documented (Paetsch et al. 2003). However, the agile methods do not cast all documentation aside (Baker 2005), and it is not necessary to write piles of documentation in order to comply with, for example, CMMI (Bos, Vriens 2004). Kähkönen and Abrahamsson (2004) did however find that XP needs some additional documentation, but it is also pointed out that the level of documentation required for a certification does not contradict the core XP principles (Wright 2003). It is recommended by the literature that the documentation is implemented as a natural part of the iterative development strategy; for example by treating both the documents and the implemented functionality as acceptable deliverables for an iteration (Namioka, Bran 2004). In this way the documents are created incrementally. The importance of adhering to the agile manifesto and still value working software over written documentation is underlined (Theunissen et al. 2003). To accomplish this, it is recommended to use a mantra that states "provide just enough documentation to be a useful reference and to help with enforcement of existing processes" (McMichael, Lombardi 2007, p. 264).

The agile and the documentation-driven methods handle the requirements differently, this can also cause problems when meshing them (Boehm, Turner 2005). The agile methods rely on user stories written by the customer in a plain business-like language. Such user stories cannot be used directly in safety critical projects. Instead each story must be translated into functional test cases (Beznosov 2003). Nawrocki et al. (2002) suggest making the tester responsible for the documentation and management of the requirements, as the tester in XP works in close collaboration with the customers and helps them write functional tests. They also recommend introducing a requirements engineering phase at the beginning of a project, which can provide a wider perspective of the system.

To ensure adaptability, agile methods advocate an iterative development strategy (Larman 2004; Nerur, Balijepally 2007), adopting agile practices requires a change in the development strategy to an iterative model and entails major alterations to work practices (Nerur et al. 2005; Senapathi 2010). In contrast to the documentation-driven methods, that have separate coding and testing phases, the agile methods include the testing and debugging of the written code in the iteration (Cohn, Ford 2003). Stålhane and Hanssen (2008) advocate that even though review meetings and the writing of documents is added to the practice, it is still possible to keep the iterations and build the software in increments.

The main asset of a software organisation is its intellectual capital; knowledge management therefore becomes very important (Lindvall, Rus 2002). Knowledge sharing is an important part of both agile and documentation-driven methods (Chau et al. 2003). While the agile methods rely on person-to-person communication and knowledge sharing, documentation-driven methods focus on documented knowledge (Chau, Maurer 2004). The frequent meetings (stand-up meetings, planning meetings and retrospectives) proposed by the agile methods serve as ways to informally communicate and share knowledge, enhanced by the self-managing developer teams. The documentation-driven methods favour division of labour, hence use role-based teams and the knowledge is shared in the documents (Chau, Maurer 2004). In the terms used by Hansen et al. (1999), the agile methods primarily rely on a personalization strategy, while the documentation-driven methods primarily rely on a codification strategy.

The agile and the documentation-driven methods have different strategies for testing. Test-driven development has become increasingly popular within the agile community (Nerur et al. 2005). In XP, testing is a core practice and development is generally test-driven (Beck, Andres 2004). It is not only suggested that test cases are written before the software, it is also more importantly suggested that all parts of an increment is tested and that completion is determined by the increment passing all tests. In documentation-driven methods testing is done late in the stages and is based on detailed test plans (Boehm, Turner 2003c).

The culture in documentation-driven organisations tends to be plan-orientated, whereas the agile cultures are more social and team orientated (Dahlberg et al. 2006), this issue can make the implementation of a meshed practice difficult. When adopting agile practices in a documentation-driven organisation, the organisational culture has to change from policy and procedure-based to freedom of development and management by team members (Misra et al. 2010). Research has described the adoption of agile methods as a culture change (McAvoy, Butler 2009). The organisational culture is very difficult to change as it requires a change in the mind-sets of people (Moe et al. 2010). Small organisations in particular struggle to adopt quality assurance, as the culture of smaller organisations, just as agile organisations, often can be characterised as creative, dynamic, and innovative (Kelly, Culleton 1999).

2.5 Meshing the Agile and the Documentation-Driven Methods

Several software organisations are trying to implement a software practice that includes elements from both agile and documentation-driven methods. As the purpose of this thesis is to study how such a practice can be implemented, a term describing such a software practice is needed. The term ‘meshed’ is chosen. Meshing is in the dictionary defined as: “to make or become entangled or entwined” (Oxford Advanced Learner's Dictionary 2011). It refers to a practice in which elements from both methods are included; the elements do not only exist together, but become entangled and entwined and a new meshed unit is formed, thus referred to as a meshed practice. Meshing shows some degree of compatibility of the agile and documentation-driven practices.

The compatibility of the agile and the documentation-driven methods and how to implement a meshed practice are discussed in the previous literature. Implementing a meshed practice in an existing software organisation can be done either by adopting agile practices in a documentation-driven software practice or by adopting documentation-driven practices (e.g. adopting quality assurance) in an agile software practice. Even though the adoption of agile software practices appears to be straightforward, introducing agile practices into a documentation-driven organisation has in many cases proven to be a challenging task (Mahanti 2004). Adoption of documentation-driven practices in an agile organisation in order to able to comply with one or several quality standards can also be a difficult task, and one which several organisations have struggled with (Ngwenyama, Nielsen 2003). The difficulties arise due to the differences between the agile and the documentation-driven software development methods (Nerur et al. 2005).

Previous research proposes several models, methods or frameworks for implementing a meshed practice. The Microsoft Solutions framework for example is an iterative and adaptive method as it is light on documentation and heavily automated through tooling, but it is still formal with respect to approvals and audits (Anderson 2005). Another suggestion is a 5-step method that uses risk analysis to define and address risks particularly associated with agile and documentation-driven methods (Boehm, Turner 2003c). Risk management offers a way to answer the question: how much is enough? (Boehm 2002). The method relies on the developer’s ability to understand their environment and to collaborate with the project stakeholders (Boehm 2002). Galal-Edeen et al. (2007) have analysed Boehm and Turners risk-based method and found its home ground concept to be practical when dealing with development projects. Taylor et al. (2006) have adapted the method developed by Boehm and Turner in order to help small software organisations adopt agile practices by assessing the risks of their organisations and managing these risks.

Since the existing software process improvement approaches are targeted at the context of the documentation-driven software development, it is advocated that organisations attempting to adopt agile practices need guidance from a framework (Pikkarainen et al. 2005; Sidky et al. 2007). Pikkarainen et al. (2005) suggest an agile deployment framework, the purpose of which is to make agile principles and practices part of an assessment process. Sidky et al. (2007) propose an agile framework consisting of two levels of assessment: one at the project level and one at the organisational level.

Some researchers engage themselves in extending the agile methods to fit with the documentation-driven practices of the quality standards. One way is by extending the agile practices, but one needs to be careful not just to add more steps and documents, as it will reduce the benefits of the agile method. A possible solution is extending the planning game of XP to include several types of user-stories. The practice is still iterative and the outputs of the iterations are to be easy to rework and do not rely too much on documentation (Boström et al. 2006). Beznosov (2003) introduces eXtreme Security Engineering, that is an adoption of agile principles and XP principles in security engineering. As with the extended XP practice proposed by Boström et al. (2006), it consists of small iterations with small releases. Zanatta and Vilain (2006) suggest an extension to the agile method Scrum, and create a method called xScrum. This method contains a set of guidelines for Scrum to conform to parts of CMMI. The Scrum philosophy is emphasised and at the same time software engineering practices to help solve problems are applied. The risk management and management issues of Scrum can also be adapted to make it compliant with the CMMI project management process areas (Marçal et al. 2008). Laurila (2004) proposes changes to CMMI in order to make it more compatible with agile ideas. An agile CMM framework, that embraces change instead of having a plan-driven focus, and that highly values people and communication is suggested.

2.6 Research Question

The research question is formulated based on the review of the literature presented in the theoretical background and based on identification of the knowledge gaps. The purpose of the research question is to provide a narrow frame for the study, create focus and guide the study and its outcome.

Previous literature has shown that the agile and the documentation-driven methods have some similarities, but also several differences. As the agile manifesto suggests, they are developed on opposing concepts. The literature identified the amount of documentation and the way the requirements are handled as the largest differences between the agile and the documentation-driven methods. In spite of these differences, practitioners experience a need for creating a software practice that consists of both agile and documentation-driven principles and practices, and several organisations are attempting to implement a meshed practice (Boehm, Turner 2003a). The review of previous literature also showed that researchers are discussing the compatibility of the agile and the documentation-driven methods; several researchers agree that despite the differences the agile and the documentation-driven methods are compatible and that it is possible to implement a meshed software practice (e.g. Baker 2005; Turner, Jain 2002).

Research conducted in this area so far does not, however, provide a detailed description of the level of compatibility; thus it is hard to determine how compatible agile and documentation-driven software development is. Due to the differences between the agile and the documentation-driven methods, many challenges are faced when trying to mesh these (Opelt, Beeson 2008). So far very little empirical data documents the alleged compatibility, and it is not well understood how these methods can be meshed in practice. The following research question has been formulated:

How can the agile and the documentation-driven methods be meshed in practice?

This research question takes a practical viewpoint of how to mesh the agile and the documentation-driven methods, in order to provide empirical evidence regarding if and how this can be done. It will be studied via case studies of the development of safety-critical software. The strict requirements of the quality standards required when developing safety-critical software makes it difficult to adopt agile practices, since the differences between agile and documentation-driven methods are heightened. Researchers are therefore discussing whether or not agile methods are suitable for such projects, at the same time more and more organisations developing safety-critical software are interested in adopting agile practices in order to increase the flexibility of their software practice (Vogel 2006); an extension of the applicability of the agile methods is therefore required (Abrahamsson et al. 2009). The documentation of the adoption of agile methods in areas of, for example, safety-critical software is sparse (Cawley et al. 2010), and due to the lack of empirical research documenting how agile and documentation-driven methods can be meshed in practice, it is not well understood how to accomplish this (Kähkönen, Abrahamsson 2004).

3. THEORETICAL FRAMEWORKS

This chapter presents the theoretical frameworks of the thesis. The research builds on the home grounds of agile and documentation-driven methods, interorganisational relationships and knowledge transfer. The framework presenting the home grounds of the agile and the documentation-driven methods provides a frame of reference for comparing these methods. The framework for interorganisational relationships provides an understanding of how adopting quality assurance is affected by the customer-vendor relationship. Agile methods rely highly on the relationship with the customer and the interorganisational relationship in these cases may therefore become even more important. Finally, the framework of knowledge transfer provides a structure for understanding potential barriers and challenges of the adoption and diffusion of agile practices in a documentation-driven organisation. This chapter explains the frameworks in relation to previous research.

3.1 The Agile and the Documentation-Driven Home Grounds

A home ground is the project characteristics within which a method performs very well (Boehm 2002). The agile and the documentation-driven methods are grounded in opposing concepts (Vinekar et al. 2006) and each method has so far been preferred for different types of project. Boehm and Turner (2003a) have identified five factors that determine the home grounds of the agile and the documentation-driven methods. These factors are: 1) size (number of personnel), 2) criticality (loss due to impact of defects), 3) culture (freedom versus procedures), 4) dynamism (the speed of which the requirements change) and 5) personnel (the skills of the personnel). These factors are used to develop the home grounds of agile and documentation-driven development; both consist of 12 characteristics divided into 4 categories: personnel, management, technical and application (Boehm, Turner 2003a) (see table 7).

While the primary goals of the agile methods are rapid value and responsiveness to change, the documentation-driven methods prioritise predictability, stability and high assurance (Boehm, Turner 2003d). The agile methods respond to change through iterative development, include customer relations, and use test-driven development and self-organising developer teams. The documentation-driven methods tend to achieve stability by developing the software in sequential phases, that include the customer in the beginning and the testing of the software as the final activity. There are also a number of differences in the way the agile and the documentation-driven methods are managed (Boehm, Turner 2005). While the agile methods depend on close contact with the customer throughout the development, the documentation-driven methods generally depend on a contract specified between the developers and the customer at the start of the project. The agile methods see planning as a means to an end rather than a means of recording in text, and use several practices to support the communications between the developers (such as pair programming and stand-up meetings). Documentation-driven methods use the documented plans to provide communication. A primary difference between the agile and the documentation-driven methods is the way they deal with the design and architecture of the software. The agile methods, in order to accommodate change, advocate a simple design that emerges as functionality is implemented, alternatively the documentation-driven methods use planning and architecture-based design to accommodate foreseeable change (Boehm, Turner 2003d). In order to be able to test the software, the code needs to be developed and executable. The short iterations with an increment as the output is the agile way of handling this issue, while the documentation-driven methods address the issue by developing and consistency-checking requirements and architecture specifications in the early stages of the development (Boehm, Turner 2003d). In an agile culture, the people feel comfortable and empowered when there is a high degree of freedom and they get to define their own work tasks. In a documentation-driven culture, the people feel comfortable and empowered when there are clear policies and procedures that define their role (Boehm, Turner 2003d). Experience shows that agile methods work best for small to medium teams; scaling up of the methods has proven difficult, mainly because of the reliance on tacit knowledge. Documentation-driven methods on the other hand scale better to large projects because the use of plans and documentations provide a better basis for communications across large groups (Boehm, Turner 2003d). Outside of these home grounds, a method that meshes the agile and the documentation-driven practices is preferable (Boehm, Turner 2003c).

Characteristics	**Agile development**	**Documentation-driven development**
Application		
Primary goals	The primary goals of agile methods are rapid value and responsiveness to change.	The primary goals of documentation-driven methods are predictability, stability, and high assurance.
Size	Agile methods seem to work best with small to medium teams of people working on relatively small applications.	Documentation-driven projects scale best to large projects (large teams and products).
Environment	Agile methods are most applicable to turbulent, highly changing environments.	Documentation-driven methods work best when the requirements are determinable in advance and remain relatively stable.
Management		
Customer relations	Agile developers use working software and customer participation to instil trust in the systems they have developed, and the expertise of their people.	Documentation-driven methods generally depend on some sort of contract between the developers and customers as the basis for customer relations.
Planning and control	Agile methods see planning as a means to an end rather than a means of recording in text.	Documentation-driven methods rely on documented plans and a project manager in charge of the control.
Communications	Agile methods rely heavily on tacit, interpersonal knowledge.	Documentation-driven methods rely heavily on explicit documented knowledge.
Technical		
Requirements	Most agile methods express requirements in terms of adjustable, informal stories.	Documentation-driven methods use formally base lined, complete, consistent, traceable, and testable specifications.
Development	Agile methods advocate a simple design, that emerges as functionality is implemented.	Documentation-driven methods rely on extensive design.
Test	Agile methods use executable test cases, that define requirements, the development is test driven.	Documentation-driven methods use documented test plans and procedures. They use late-testing.
Personnel		
Customers	Agile methods strongly depend on dedicated, collocated customers.	Documentation-driven methods count on up-front, customer-developer work on contractual plans and specifications.
Developers	Critical people-factors for agile methods include amicability, talent, skill, and communication.	Documentation-driven methods of course do better with great people, but are generally more able to plan the project and architecture of the software so that less-capable people can contribute with low risk.
Culture	In an agile culture, the people feel comfortable and empowered when there is a high degree of freedom.	In a documentation-driven culture, the people feel comfortable and empowered when there are policies and procedures that define their role in the enterprise. Each person's tasks are well-defined.

Table 7. The home grounds of the agile and the documentation-driven software methods, adapted from Boehm and Turner (Boehm 2002; Boehm, Turner 2003c; Boehm, Turner 2003a; Boehm, Turner 2003d)**.**

The home grounds were developed in order to guide software developers in their choice of an appropriate software development method (Boehm, Turner 2003c). As the home grounds compare the general characteristics of agile and documentation-driven software development, they are also useful for analysing the characteristics of a specific software practice and determine whether and how it meshes the practices and principles of the agile and the documentation-driven methods.

3.2 Interorganisational Relationships: Attributes and Processes

Previous research has concluded that interorganisational relationships in general have a great influence on the outcome of the cooperation between a vendor and a customer (Das, Teng 1998; Holmström Olsson et al. 2008; Kern 1997). The agile software methods advocate and depend highly on close and well-functioning cooperation between the software developers and the customer; XP for example introduces the idea of an onsite customer (Beck, Andres 2004). In these cases the interorganisational relationship becomes even more important. The adoption of quality assurance in an agile software development practice involves both the vendor and the customer; the interorganisational relationship is therefore likely to have a great influence on this. Adopting quality assurance in an agile software practice is difficult and many IT companies have and are struggling with this (Batista, Figueiredo 2000). A lot of research has been done on small software companies trying to implement software process improvement practices in terms of quality standards like the CMMI (e.g. Batista, Figueiredo 2000; Brodman, Johnson 1994; Tjørnehøj 2006). But, no research has addressed how interorganisational relationships influence the adoption of quality assurance in an agile software organisation.

The framework primarily builds on the processes and attributes of a successful interorganisational relationship identified by Goles and Chin (2005). Additionally, the literature also revealed that trust is an important attribute, and thus it has been added to framework in the list of attributes. The attributes and processes are summarised in table 8 and table 9, and are described in more detail below.

Attributes	Description
Interdependence	Interdependence is "the extent to which each party of the interorganisational relationship is dependent on the other party in order to achieve its goals." It suggests that the parties have complimentary assets and skills and that they need each other to achieve their goals.
Flexibility	Flexibility can be defined as "the willingness of both parties to make adaptations as circumstances change." It has proven fundamental to a relationship; if a relationship does not have the capacity to accommodate change it is likely to break up when the necessity of change exerts pressure.
Cultural compatibility	Cultural compatibility is "the extent to which each party can coexist with the others' beliefs and values." Differences in organisational culture can cause difficulties in a relationship; minimising the cultural differences can advance the process of achieving shared objectives.
Commitment	Commitment is "the willingness of the parties of an interorganisational relationship to exert effort and devote resources in order to sustain an ongoing relationship." It has been found to be a major contributor to sustaining a relationship.
Consensus	Consensus is "the extent of general agreement between parties." It has been found to assist in the coordination and implementation of decisions.
Trust	Trust has been defined as: "the expectation that a party will act predictably, will fulfil its obligations, and will behave fairly even when the possibility for opportunism is present." Trust has great influence on a relationship, it for example allows the parties to focus on long-term goals, they worry less about day-to-day issues, it enables risk-taking and it reduces conflict.

Table 8. The attributes of a successful interorganisational relationship, adapted from (Goles, Chin 2005)

Processes	Description
Cooperation	Cooperation is defined as "the undertaking of complimentary activities to achieve mutual benefits." Cooperation is a higher-level abstraction to the concept of coordination.
Communication	Communication is: "proactive formal and informal sharing or exchange of meaningful and timely information between parties."
Conflict resolution	Conflict Resolution is "the amicably replacing disagreement with agreement." Conflict in a relationship is inevitable, but the way it is resolved has implications for relationship success.
Coordination	Coordination is about "managing interdependencies between parties." Research argues that successful relationships are characterised by coordinated actions.
Integration	Integration is "intertwining attributes and processes of the relationship into each party's structure, policies, and procedures." Integration has proven to contribute both to the quality of interorganisational business processes and the quality of internal business processes.

Table 9. The processes of a successful interorganisational relationship, adapted from (Goles, Chin 2005)

Through a thorough literature review of previous research, Goles and Chin (2005) constructed a list of factors that play a significant role in relationships, and grouped these into attributes and processes. The five attributes that contribute to the functionality and harmony of the relationship are commitment, consensus, cultural compatibility, flexibility and interdependence and the five processes that develop the attributes are communication, conflict resolution, coordination, cooperation and integration. Reviewing other relevant studies support the conclusion that these attributes and processes are those most significant for a successful relationship. Using the attributes and processes as a framework, Holmström Olsson et al. (2008) conducted a case study of an offshore sourcing relationship. They concluded that the attributes and processes were valuable for initiating a discussion with their interviewees, to get the interviewees to reflect on the relationship, and that the attributes and processes were valuable in order to understand how such a relationship can be understood.

Besides the processes and attributes of Goles and Chin (2005), trust has also been proven to play an important role in achieving a successful relationship (Sabherwal 1999). Trust is both an important factor in interpersonal and in interorganisational relationships (Sabherwal 1999) and the study of Zaheer et al. (1998) showed that in an interorganisational relationship trust clearly matters. Trust can be defined as "positive expectations about another's motives with respect to oneself in situations entailing risk" (Das, Teng 1998, p. 494). Trust, commitment and communication are highly dependent; commitment and communication leads to greater trust and vice versa (Kern 1997). Goodwill trust is trust in one's good faith, good intentions and integrity, whereas competence trust is the trust in one's ability to do appropriate things, and not trust in the other party's intention to do so. Having goodwill trust in a relationship, the partner will be less concerned with problems of cooperation (Das, Teng 2001). It is noticeable that trust is hard to build, but easy to violate (Brunard, Kleiner 1994).

Quality assurance is a formal control mechanism (Das, Teng 1998). Research has pointed out that trust and control influence each other; for example both formal and social control can be successfully implemented in an interorganisational relationship, but may have different impact on the level of trust (Das, Teng 1998). Formal control mechanisms can be a hindrance to a high level of trust, through their constraining of people's autonomy, whereas social control as a by-product can be trust-building as it often takes the form of socialising and interaction (Das, Teng 1998). Complete trust in a partner could point to no or a very low need for control mechanisms, but exclusive reliance on trust in relationships introduces an increased risk of failure, as structures or potential problems will be disregarded. Relying on excessive structural control can, however, hurt performance. Most projects therefore need both trust and control (Sabherwal 1999).

3.3 Knowledge Transfer

Knowledge transfer can be defined as: "the process through which one unit (e.g. group, department, or division) is affected by the experience of another" (Argote, Ingram 2000, p. 151). Research has defined two types of knowledge: tacit knowledge, which is deeply rooted in ones actions, and explicit knowledge, which is transmittable in a formal, systematic language (Nonaka 1994). In systems development, transferring knowledge from one project to another is very difficult (Lyytinen, Robey 1999), mainly because much of the knowledge about software engineering is tacit (Desouza 2003). Besides from the sharing activities, knowledge transfer also includes implementing and integrating the new knowledge in ones practice. To fully understand knowledge transfer in agile practices, it is relevant to include literature on the adoption of agile practices as well as general literature on knowledge transfer.

The framework primarily builds on the barriers identified by Riege (2005). The development of the framework has furthermore been supported by a review of the literature on knowledge transfer barriers and on the literature of the adoption of agile methods. The potential barriers have been divided into seven categories: individual skills, motivation and willingness, time and resources, organisational culture, management style, trust and infrastructure. The barriers are summarised in table 10 and described in more detail below.

Barrier	Description
Individual skills	Individuals and their skills are central to the outcome of a knowledge transfer process.
Motivation and willingness	The individuals may be more or less motivated to receive knowledge.
Time and resources	Time is a significant factor, as the process of identifying and transferring knowledge is very time-consuming.
Organisational culture	Culture is often seen as the key inhibitor of effective knowledge transfer. Culture also plays a major role when adopting agile software practices, as the entire organisation is affected.
Management style	The adoption of agile practices in a documentation-driven organisation requires a change in management styles.
Trust	Without trust people are unlikely to share knowledge.
Infrastructure	Integrating an IT system can support the knowledge transfer process.

Table 10. The barriers to the knowledge transfer of agile practices

Individual skills have proven to have great influence, without the right skills knowledge is not likely to be transferred. Communication skills are for example extremely vital (Riege 2005). Being part of a social network is also important in order to be able to draw on other people's knowledge (O'Dell et al. 1998). Differences, such as age, gender, educational level and ethnic background can also have great influence (Tellez-Morales 2009). Another potential barrier is the absorptive capability of the individual; this varies and is a boundary to knowledge transfer (Lyytinen, Robey 1999). Agile methods rely heavily on human factors, and these are therefore vital to the adoption of agile practices (Esfahani et al. 2010).

Individuals can be more or less motivated and willing to receive knowledge (Szulanski 2000) and they need to see a value of the knowledge in order to share their knowledge voluntarily, knowledge transfer cannot be forced (Riege 2005). Getting people to change their practices is very difficult as they tend to fall back on their routines (Gupta 2008).

The process of transferring knowledge is very time-consuming, time and resources therefore become central factors. A lack of time can result in the knowledge transfer process only partially succeeding, and a lack of resources to create the opportunities for knowledge transfer can also be a barrier to the process (Riege 2005).

Organisational culture is hard to change, it is often seen as the key inhibitor of knowledge transfer in the literature on knowledge management (McDermott, O'Dell 2001). When adopting agile practices, the organisational culture needs to change from one of policy and procedure-based to one of freedom of development and management by team members (Misra et al. 2010).

The adoption of agile practices may require a change in the management style (Boehm, Turner 2005). In a documentation-driven method, the management style is command-and-control, while the agile methods operate with a leadership-and-collaboration management style (Misra et al. 2010). In order to transfer knowledge it is important that the organisation and its manager prioritise knowledge retention (Riege 2005).

A lack of trust can be a barrier; if people do not trust each other they are not likely to offer important knowledge and will be reluctant to accept knowledge. The level of trust in an organisation, between its sub-units and its employees seems to have a direct influence on the communication flow and thus the amount of knowledge transferred; it is mostly in informal networks that people trust each other, voluntarily share knowledge and insights with each other (Riege 2005).

Integrating an IT system can support the knowledge transfer process, such a system however requires an adequate level of informational and technical infrastructure (including support and maintenance). It further needs to support the practices of the organisation, otherwise actors may be reluctant to use it (Riege 2005).

4. RESEARCH APPROACH

The research approach will be presented in this chapter. It includes a description and argumentation for the overall research design, that includes a literature review and two interpretive case studies. Furthermore, the research approach of the literature review and the case studies will be described in detail.

4.1 Research Design

The choice of research design is guided by the purpose of the study and the research question. The purpose of this study is to understand how the agile and the documentation-driven methods can be meshed in practice. For this purpose the research design consists of a literature review and two case studies; the research design logic is depicted in figure 2. Reviewing previous literature related to the research topic at hand is an essential first step and foundation of a research project (Baker 2000). The purpose of the literature review was to obtain an overview of the existing literature on the topic and be able to identify important, critical knowledge gaps. The literature review therefore serves as a theoretical foundation for the study. In order to answer the research question, empirical data is needed as well. To provide this empirical knowledge, two interpretive case studies are conducted (see details in section 4.3). The first case study is of a small, agile software organisation and the second case study involves a large pharmaceutical organisation. Both organisations have engaged themselves in developing a safety-critical, pharmaceutical device. These cases are highly interesting and suitable for this study; the strict quality assurance required by safety-critical standards such as FDA and GAMP highlights the differences between the documentation-driven method and the agile method. Two case studies are chosen, as it gives the possibility of a comparison and increases the possibility of generalising. The literature review and the case studies are conducted within the area of concern (meshing agile and documentation-driven methods in practice), and result in contributions that offer new knowledge.

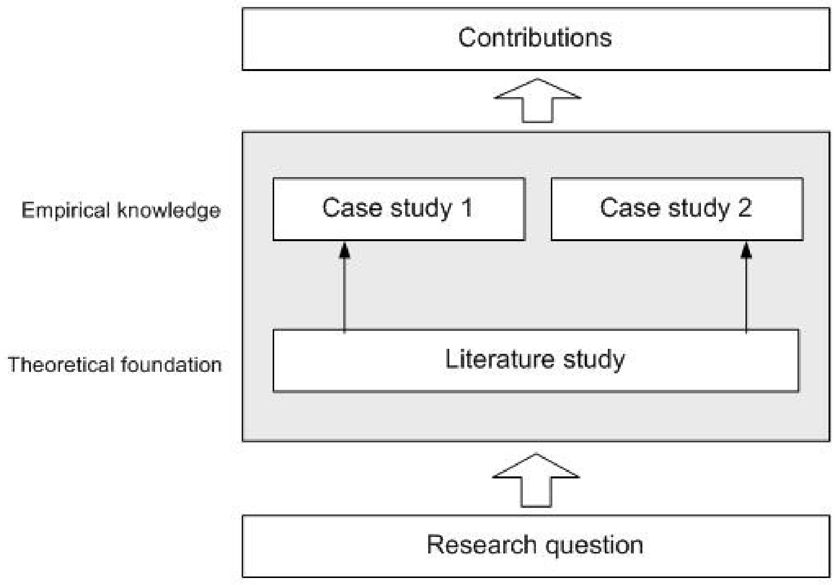


Figure 2. The research design logic

A research project consists of three elements, the area of concern (A), the method (M_R) used for studying A and the framework (F) that provides a focus to the phenomenon studied in A (Checkland, Scholes 1990). Figure 3 summarises the area of concern, the research method and the frameworks of this thesis. The area of concern (A) in this thesis is how to mesh the agile and the documentation-driven methods in practice when developing software (see research question, section 2.6). The research method (M_R) used includes both a literature review and two interpretive case studies. The frameworks (F) used for studying the area of concern are: the home grounds of the agile and the documentation-driven methods (section 3.1), the attributes and processes of interorganisational relationships (section 3.2) and the barriers to knowledge transfer (section 3.3). The frameworks provide different perspectives on the use of agile and documentation-driven software development in practice.

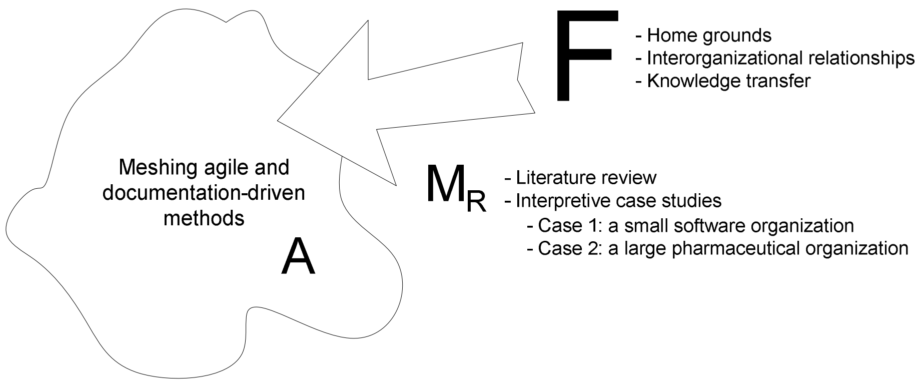

F
- Home grounds
- Interorganizational relationships
- Knowledge transfer
Meshing agile and documentation-driven methods
A
M R
- Literature review
- Interpretive case studies
- Case 1: a small software organization
- Case 2: a large pharmaceutical organization

Figure 3. The three research elements, adapted from Checkland and Scholes (1990)

The Research Activities

Figure 2 and 3 do not indicate the order of the activities of the study. Figure 4, on the other hand, shows the timeline of the research and depicts the activities of the research design in the order they were carried out. The two case studies were conducted sequentially; making it possible to focus on one study at the time. The literature review was done simultaneously with the two case studies, in order to identify relevant theories for the analysis of the data, providing both the body of knowledge and the analysis frameworks. Together, the research activities contribute to the theoretical and empirical contributions; these were documented in five research papers (see chapter 5).

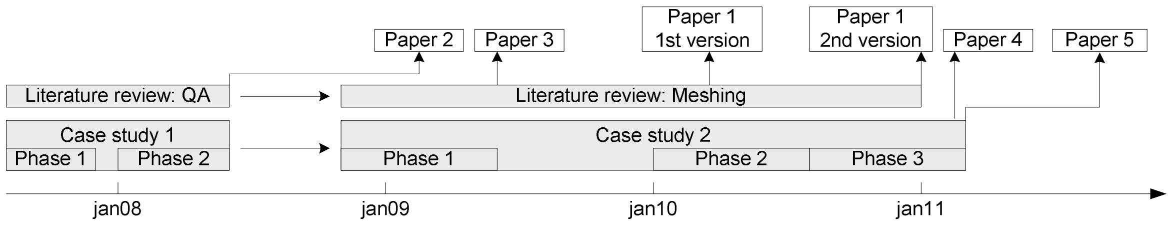


Figure 4. The timeline of the research

The research was initiated with case study 1. This study consisted of two phases of interviews, observations and document studies. The interviews of the first phase focused on the software organisation and the challenges of meshing. The analysis of the first phase showed that to fully understand the situation at hand a broadened perspective that included the customer relation was needed. Hence, phase 2 included interviews with developers from the software organisation and with representatives from their customer. The research approach of case study 1 is expanded in section 4.3.

Case study 2 consisted of three phases focusing on two projects within the same organisation. The two first phases focused on software group A. In phase 1 the focus was on mapping the software practice of software group A, at this point they had just started meshing. In order to get data on how software group A improved their software practice and increased the mesh, a second phase of data collection was conducted. After finishing the analysis of software group A, the focus was broadened to include software group B as well, in order to increase the understanding of the barriers and challenges of meshing. The two software groups were working close together and knowledge on how to mesh was transferred from software group A to software group B. Including software group B in the study could therefore offer more knowledge on how a mesh could be achieved. Software group B was determined to overcome the barriers identified in the analysis of the mesh in software group A. The research approach of case study 2 is expanded in section 4.3.

4.2 Literature Review

According to Webster and Watson (2002), the primary goal of a literature review is to create a map of the existing literature based on the key concepts. In order to accomplish this, the main activities are 1) identifying relevant literature and 2) structuring the review. A literature review based on a strong collection of existing literature and on a structured analysis is more likely to result in a strong synthesis and evaluation.

Identification of Relevant Literature

To ensure an accumulation of a relatively complete consensus of literature, the search was conducted in three steps, including a search of the web, a search through relevant journals and conferences and a backward and a forward search of the papers identified (Webster, Watson 2002).

First, the web (using Google Scholar) was searched using relevant keywords. The keywords contained combinations of documentation-driven and agile methods and more general keywords such as ‘mesh and agile’. In parallel, ranked journals and relevant conferences were searched. The search of ranked journals was conducted as the major contributions are most likely to be in leading journals (Webster, Watson 2002). The scarce number of papers identified in these journals shows that much research on meshing agile and documentation-driven methods are still to be done, and that the major contributions are still missing. Secondly, a backward search (reviewing the citations of the papers identified) and a forward search (using Google Scholar to identify papers citing the papers identified) were carried out. This step was done iteratively. The potentially relevant papers identified through step 1 and step 2 was found relevant to the research question based on their title and abstract (Baker 2000). At this point the search resulted in 88 papers.

Finally, all 88 papers were read, nine of these were discarded as a closer look revealed a lack of relevance. In some of the papers, the compatibility of the agile and the documentation-driven methods was a very small and insignificant part of the paper, while other papers only dealt with either the agile or the documentation-driven methods and not both. The final selection resulted in 79 papers.

As Information Systems is an interdisciplinary field, it is also recommended that a search includes papers outside the field (Webster, Watson 2002). The search of the web and the conduction of the backward and the forward search were done in order to identify potentially relevant papers outside the Information Systems field.

Structuring the Review

Having identified the papers with direct relevance, the key arguments of each paper must be analysed (Baker 2000). Bandera et al. (2011) present a systematic tool-supported method for conducting literature reviews. In their method, the analysis of the selected papers was structured by a computer program for qualitative data analysis called NVIVO. To structure the analysis of this literature review Atlas.ti V6 (Muhr 1991), a computer program similar to NVIVO, was used. The coding consisted of three linear steps. 1) To identify the key concepts in the papers, the first step was coding the papers using open coding (Strauss, Corbin 1990). The open coding was however adapted as the overall purpose of the literature review at this point had already been pre-determined and the focus when reading and coding the papers was therefore on the compatibility of the agile and the documentation-driven methods, challenges of meshing, and strengths and weaknesses of a meshed practice. The first step resulted in 59 different codes applied to 1038 quotations. 2) The second step of the coding was to categorise the codes, thus creating a structure to the analysis by creating coding schemes containing the key concepts identified. The coding schemes served as the analysis frameworks. In the third and final step the codes and quotations from step 1 were revisited and recoded according to the coding schemes. This resulted in 71 quotes applied to 1084 quotations.

4.3 Case Studies

To ensure a detailed understanding of the software practices, each case study was conducted over a longer time period of approximately 1-3 years, and a significant amount of data were collected: 47 qualitative interviews, several observations and document studies. Table 11 gives an overview of the two interpretive case studies and their phases.

#	Case	Duration	Data collection	Focus
1	A small, agile software organisation	Phase 1: August 2007 – December 2007	4 qualitative interviews, observations and a document study	The adoption of quality assurance at the software organisation
		Phase 2: January 2008 – June 2008	5 qualitative interviews, and a document study	The influence of the customer-vendor relationship on adoption of QA
2	A large pharmaceutical organisation	Phase 1: November 2008 – June 2009	15 qualitative interviews, observations and a document study	The software practice
		Phase 2: January 2010 – August 2010	12 qualitative interviews, observations and a document study	The software practice and its changes/improvements
		Phase 3: August 2010 – March 2011	11 qualitative interviews and a document study	The knowledge transfer of agile practices between software teams

Table 11. Overview of the case studies

Qualitative interviews have been recognised as one of the most important data gathering tools in qualitative research (Myers, Newman 2007). It is through the interviews that the researcher has the best opportunity to access the interviewee’s interpretations of the actions and events, that have or are taking place (Walsham 1995). The qualitative interview was therefore chosen as the primary data collection technique. The interviews were organised as semi-structured interviews (Myers, Newman 2007). The interview guides were prepared beforehand to ensure the structure and relevance of the interviews; the semi-structured guides however also gave room for the investigation of other interesting subjects that arose. Both case studies were initiated by a problem diagnosis initiative, as performing an assessment of the software development practice gives an understanding of the current software practice and its strengths and weaknesses (Iversen et al. 1999).

As interviews are social interactions they are fraught with difficulties; examples are: ‘artificiality of the interview’, ‘lack of trust’, ‘lack of time’ and ‘construction of knowledge’ (Myers, Newman 2007). Interviews are an interrogation between strangers, and the interviewer is asking the interviewees to create opinions under pressure of time. As the interviewer and the interviewees are strangers, the level of trust between them might not be very high and can influence the answers. In interpretive case studies it is acknowledged that the interviewer is human and interprets the responses – constructing knowledge in the present.

In order to overcome or at least minimise these challenges/risks, some of the recommendations given by Myers and Newman (2007) were used. They advise interviewers to use mirroring in questions and answers; that is taking the words and/or phrases of the interviewees and using these in constructing a follow up question. In this way the interviewees are prompted to use their own language. Confidentiality of disclosures is another recommendation, it involves assuring the interviewee that the recordings and transcriptions will be kept confidential and secure (and of course do it); this helps create mutual trust. A third recommendation is to represent various voices; making sure to interview a variety of people (triangulation of subjects). The interviewees included in the case studies of this thesis are therefore at different organisational levels, they have different work tasks and different prior experiences.

All of the interviews and parts of the observations were audio recorded and transcribed for analysis. Recording of the interviews can make the interviewee uncomfortable, but the advantage of a recording is that it is possible to make a transcript of the interview and return to the transcript later for alternative forms of analysis (Walsham 2006).

Interviews should be supplemented by other forms of empirical data in an interpretive study (Walsham 2006). It is recommended to use a triangulated approach in order to gather different types of data, that can be used as cross checks; the aim of triangulating the data collection is to draw on the particular and different strengths of various empirical data (Pettigrew 1990). Other data can be internal documents, direct observations or participant observations of actions, and web-based data from for example emails and websites (Walsham 2006). While interviews can provide depth, subtlety, and personal feeling, interviews may also be staged occasions where feeling and evocation is high and factual detail low. Documents can provide facts, but are subject to dangers of selective deposit and survival. Observations provides access to group activities and can confront the researcher with discrepancies between what people have said in interviews and what they actually do (Pettigrew 1990). This is why the interviews of the case studies have been supported by both observations and document studies.

Pettigrew (1990) has portrayed research as an iterative process, that includes the following phases: observation, induction, deduction, verification, and further observation. The data collection and data analysis of the case studies were done in an iterative manner, and each case study is divided into 2-3 phases, each with a separate focus. The following subsections describe the research approach of each case study in more detail.

Case Study 1: The Small Software Organisation

This is an interpretive case study (Walsham 1995) of the process and challenges of adopting quality assurance in a small, agile software organisation in Denmark. The case study focuses on the adoption of the GAMP standard by a small, agile software organisation developing pharmaceutical software for a customer in the public sector. The case was chosen as the software organisation at the beginning of the research study expressed great frustration because of the requirements of the quality assurance standard. One of their primary goals was to maintain the agility of their software practice while complying with the GAMP standard. The customer concerned was of great importance to the software organisation – the reason why they were so eager to change their practices.

The customer had geographically distributed user sites, each with separate management, but the cooperation with the software organisation was organised and managed through a central department. The project management of the software organisation worked closely with this department in describing system requirements, while a cross-organisational steering committee prioritised the development resources.

The system development was initiated in 1998 and in October 2004, the software organisation decided to initiate the adoption of the quality assurance as they realised that the customer would soon make a demand for quality assurance. The GAMP4 standard was suggested by the customer organisation later the same year, and the quality assurance adoption was officially initialised. Two years later, in December 2007, the software organisation went through its first external audit. They failed this audit.

Data Collection and Data Analysis

The data was collected and analysed in two phases in order to gain a rich history on which the data analysis could be based. The phases of the case study are summarised in table 12.

Phase	Data collection	Data documentation	Data analysis
1	4 qualitative interviews, a document study, four months of observations and a quality seminar	Interviews audio recorded and transcribed	Using Problem Diagnosis to identify challenges/barriers to the adoption of quality assurance
2	5 qualitative interviews and a document study	Interviews audio recorded and transcribed	Using the framework of interorganisational relationships

Table 12. The phases of case study 1

The purpose of the first phase was to gain an initial understanding of the adoption of quality assurance of the software organisation, the adoption of the GAMP standard was therefore studied from the viewpoint of the software organisation, and the data collection and analysis focused on the internal practice. Four diagnostic interviews (Iversen et al. 1999) were conducted, these included employees from the software organisation: the project manager and three developers. Furthermore, one researcher made observations on the software organisation over four months, building personal knowledge of the case. At the end of the first phase a quality seminar was given by the researcher in order to prepare the software organisation for the upcoming audit. This seminar included all of the employees from the software organisation. The initial data analysis showed that studying the adoption of quality assurance from only the side of the software organisation did not reveal a full picture of the situation.

The level of focus was therefore shifted in the second phase and the adoption was studied from an interorganisational perspective. Five qualitative interviews based on semi-structured interview guides developed from the framework of interorganisational relationship (Goles, Chin 2005) were conducted. These interviews included: the project manager from the software organisation, a developer from the software organisation, two users of the system and a customer representative.

During the two phases, all interviews were recorded and transcribed. The data was analysed iteratively both in-between the phases and after the second phase. First, a historical map of important events in the adoption was compiled by working through the data and discussing them with the project manager of the software organisation. Second, the customer–vendor relationship was analysed according to the framework of attributes and processes of interorganisational relationships (Section 3.2, Goles, Chin 2005). The findings were written up and presented to the interviewees in a report.

Case Study 2: The Large Software Organisation

This is a longitudinal and interpretive case study (Pettigrew 1990; Walsham 1995) of how clinical products with embedded software are developed in a large pharmaceutical organisation in Denmark. The focus has been on two software groups from two different projects. Both software groups were developing software for a clinical device for dispensing drugs. In both projects the development also involved pharmaceutical and clinical researchers responsible for the drugs, process engineers responsible for production facilities, mechanical engineers responsible for the product's mechanical functions in dispensing the drugs, and embedded hardware engineers responsible for the computer chips and the communication controlling the mechanics. The software groups were working in close cooperation and were dependent on the engineers developing the hardware, the mechanics and the system engineering group. Due to the high level of safety-criticality of the devices, the projects had to comply with several product and process standards (e.g. FDA).

The two teams were different in size and complexity; the task of software group A was larger and more complex than the task of software group B. While software group A consisted of approximately 30 managers, developers and testers, software group B only consisted of three developers, one of these in the role of the software coordinator and Scrum master. Project A had run for several years and was entering the stage of refining the product. At the time of study project B had been focusing on developing prototypes. The first focus of this case study was on the software group of project A and how they had implemented a software development practice that meshed a documentation-driven FDA-compliant practice and an agile practice. This software practice consisted of iterations, that varied in length but settled on 2 weeks. Each iteration was initiated with a planning meeting, that included a retrospective of the previous iteration and planning of the future iteration. The developers were divided into two sub teams, that were coordinated by the Scrum master. The testers were organised in their own sub team controlled by a test manager. The two sub teams of developers worked on the same product backlog. The tasks of the iterations were displayed on two Scrum boards on the back wall in the office. The developers primarily coordinated within the sub teams through the daily stand-up meetings. The developers were responsible for unit testing and reviewing the code during each iteration; the test team conducted the integration tests before the code was handed to the system engineering group that ran the system tests of the increments, released every second month. Software group A had gained experience in developing software architecture, conducting software testing, peer reviewing and writing documentation. They had furthermore experimented with different tools to support the development practice and been through the process of having these tools validated by various quality standards. The second focus of this case study was on a knowledge transfer process between software group A and software group B. The pharmaceutical organisation initiated a knowledge transfer process between these two groups, in order to pass on the experience of software development and disseminate this knowledge within the organisation. The focus of the knowledge transfer process was the diffusion of agile practices. The knowledge and experience of software group A was transferred by several means: experience workshops, facilitation, consultancy and adaptation.

Data Collection and Analysis

The empirical data was collected and analysed iteratively through three phases, see table 13. While phases 1 and 2 focused on the software practice of software group A, phase 3 focused on the knowledge transfer between software group A and software group B. Each phase is described in detail below.

Phase	Sub Phase	Data collection	Data documentation	Data analysis
1	a	7 qualitative interviews	Interviews audio recorded and transcribed, the seminar audio recorded	Using Problem Diagnosis based on Soft Systems Methodology
	b	8 (4+4) qualitative interviews		
	c	Seminar presentation and validation		
2	a	7 qualitative interviews	Interviews and observations audio recorded and transcribed	Using the framework of the home grounds of the agile and the documentation-driven methods
	b	5 qualitative interviews		
	c	Observations of planning meetings and stand-up meetings		
3	a	1 qualitative group interview	Interviews audio recorded and transcribed	Using the framework of barriers of knowledge transfer
	b	5 qualitative interviews		
	c	5 qualitative interviews		

Table 13. The phases of case study 2

Phase 1 - The Software Practice of Software Group A: The first phase included two sub phases of interviews (1a + 1b) and a sub phase with a seminar presentation and validation of the results to date (1c). All interviews were organised as diagnostic interviews (Iversen et al. 1999).

In the first sub phase (1a), 7 qualitative interviews were conducted with the project manager, the software architect, a software tester and four developers from both sub teams. The semi-structured interview guide included the subjects: FDA requirements, agile software development, the requirement specifications, handling of errors and general strengths and weaknesses. The analysis was performed using Soft Systems Methodology (SMM) (Checkland, Scholes 1990) by (i) drawing rich pictures of the situation and problems encountered; (ii) creating root definitions and conceptual models of activity systems; and (iii) a comparison between the models and the interviews.

The second sub phase (1b) included eight interviews with the project manager, the software architect, a tester and five developers from both sub teams. Four of these interviewees participated in the first sub phase, whereas the last four were participating for the first time. Two interview guides were used. Interviewees participating for the second time were asked questions based on the activities from the conceptual models and concluded by validating some of the statements from the first interviews. New interviewees were asked questions that included the subjects of FDA requirements and agile development before discussing the activities and evaluating the statements. The data was again analysed according to SSM.

At the end of the first phase the findings were written into a report and presented to software group A in a seminar (sub phase c), including a presentation and discussion. The interviewees from the previous two sub phases participated. Besides the validation of the results, such seminars offer an opportunity for further data collection (Pettigrew 1990).

Phase 2 - Improvements of the Software Practice of Software Group A: The second phase also included two sub phases (2a + 2b) of interviews and added a sub phase (2c) of observations in order to verify the procedures and outcomes of the planning meetings and stand-up meetings. The data collection and data analysis were based on the framework of the home grounds of the agile and the documentation-driven methods (Section 3.1, Boehm 2002). The interviews were semi-structured by an interview guide based on the characteristics of the home grounds and followed up on the evaluation from phase 1.

The first sub iteration (2a) included seven interviews with 3 software developers, a tester, the new software architect, a coordinator between the software group and other groups of the project and a consultant specialising in Scrum, who had been affiliated to the software group for 5 months. The output of the interviews was an identification and description of activities within the characteristics of the home grounds and a revised evaluation. The second sub phase (2b) focused on gaining a detailed description of the activities identified in sub phase 2a. It included 5 interviewees: the former project manager, who was reassigned as being responsible for the processes, the former architect, now functioning as a software developer, the new Scrum master, a tester and a software developer. The purpose of sub iteration (2c) was to determine the procedures and the outcomes of the planning meetings and stand up meetings. It included 2 observations: 1 observation of a planning meeting and an observation of a stand up meeting.

In the end of this phase all interviews (from both phase 1 and phase 2) were collected into one hermeneutic unit and coded using ATLAS.ti V6 (Muhr 1991). The interviews were coded according to the framework of the home grounds of the agile and the documentation-driven methods, including the characteristics: agile or documentation-driven personnel characteristics, agile or documentation-driven management characteristics, agile or documentation-driven technical characteristics and agile or documentation-driven application characteristics. The analysis was supported by document studies and the observations. The full analysis was written up, presented in a report and validated by software group A.

Phase 3 - Knowledge Transfer between Software Group A and Software Group B: This phase focused on the knowledge transfer of agile practices between software group A, studied in phase 1 and phase 2 and software group B of the same organisation. The third phase also included three sub phases of data collection and analysis.

The first sub phase (3a) simply involved an initial group interview (Myers, Newman 2007) with the focus of gaining an overview of the status of software group B and the knowledge transfer process. The interview included the software coordinator of software group B and the coordinator between the software and hardware groups of project B.

The second and third sub phases (3b + 3c) included interviews with the two full time software developers, interviews with the process manager from software group A and follow up interviews with the software coordinator of software group B and the coordinator between the software and hardware in project B. The interview guides focused on the knowledge transfer process, which knowledge mechanisms were used, an evaluation of the outcomes and on the barriers hindering the implementation of the transferred knowledge about the software development practice.

The transcriptions of the interviews were coded using Atlas.ti V.6 (Muhr 1991). The analysis mapped the specific barriers hindering the knowledge transfer process, after which the barriers were categorised according to the framework of the knowledge transfer of agile practices. The results were written up in a report and validated by software group B and the process manager of software group A.

5. RESEARCH SUMMARY

This chapter presents the five research papers; it summarises each paper and describes how they complement each other by their differences in theory, empirical data, and foci.

5.1 Overview of the Papers

Table 14 provides the publication details for each paper included in the appendix: 1) the first column indicates the number of the paper, 2) the second column contains the title of the paper, 3) the third column identifies the authors of the paper, 4) the fourth column outlines the research approach of the paper, 5) the fifth column gives an overview of the data analysis and 6) the last column shows the status of the paper.

#	Title	Authors	Research approach	Data analysis	Status
1	The Agile and the Disciplined Software Approaches: Combinable or Just Compatible?	Heeager L.T.	Literature review	Based on definitions of combinability and compatibility	Presented at ISD2011
2	Adopting Quality Assurance Technology in Customer-vendor Relationships: A Case Study of how Organizational Relationships Influence the Process	Heeager L.T., Tjørnehøj G.	Case study	Based on the framework of interorganisational relationships	Published in the proceedings of ISD2009
3	Agile Software Development and its Compatibility with a Document-Driven Approach? A Case Study	Heeager L.T., Nielsen P.A.	Case study	Based on conceptual models built using SSM	Published in the proceedings of ACIS2009
4	Introducing Agile Practices in a Documentation-Driven Software Development Process: A Case Study	Heeager L.T.	Case study	Based on the framework of the agile and the documentation-driven home grounds	Resubmitted for JITCAR; status: accepted with minor revisions
5	Barriers to Adoption of Agile Development Practices: A Knowledge Transfer Framework	Heeager L.T., Nielsen P.A.	Case study	Based on the framework of the barriers to knowledge transfer	Submitted for ISJ: status: revise and resubmit

Table 14. Overview of the papers of the thesis

Collectively the five papers give a detailed understanding of how the agile and the documentation-driven methods can be meshed; each paper contributes with different angles to this issue. The first paper is a literature review that gives an overview of the compatibility and the challenges of meshing the agile and the documentation-driven methods based on the previous literature. It is used to create a foundation on which the empirical data can be collected and analysed. The main purposes of the second and third papers are to determine the compatibility of the agile and the documentation-driven software methods in practice through empirical studies, and to determine the challenges of implementing a meshed practice. Whilst the second paper focuses on implementing documentation-driven elements in an agile software practice, the third paper focuses on implementing agile elements in a documentation-driven software practice. The fourth and fifth papers build on the knowledge of the first three papers. The main purpose of paper four is to extend the knowledge on the challenges of implementing a meshed practice, hence it is a longitudinal case study of how a software team during a time period of 2 years increases the agility of its software practice so that it transforms from being primarily documentation-driven to become a more meshed practice. The fifth paper focuses on potential barriers when transferring agile practices between organisational units, in order to increase the knowledge of how to disseminate the knowledge of meshed practices in organisations. Each paper is described in more detail below.

5.2 The Agile and the Disciplined Software Approaches: Combinable or Just Compatible?

This is a literature review, the main purpose of which is to determine how compatible the agile and the documentation-driven software development are. In order to measure the degree of compatibility the paper takes it's starting point in the definitions of the two terms combinability and compatibility (Oxford Advanced Learner's Dictionary 2011). The paper defines combinability between the agile and the documentation-driven methods as: "The agile and the disciplined software approaches are combinable if it is possible to use an agile software development approach and at the same time comply with a quality assurance standard without compromising the agility of the agile approach". The compatibility between the agile and the documentation-driven methods is defined as: "The agile and the disciplined software development approaches are compatible if it is possible to use some of the agile practices and principles and at the same time be able to comply with a quality assurance standard". The second purpose of the paper is to determine the main challenges of meshing agile and documentation-driven software development.

The literature review consists of 79 papers identified through a comprehensive, iterative search, including relevant keywords, relevant journals and conference proceedings and a forward and backward search. The analysis of the papers was structured using Atlas.ti V.6. (Muhr 1991), the papers were coded through two iterations, first using a strategy similar to open coding (Strauss, Corbin 1990) followed by a categorisation of the code, hence creating coding schemes to which the coding was adapted.

The review showed that the agile and the documentation-driven development are compatible, but not combinable; it is possible to implement agile practices and principles in a development practice compliant with a quality standard, but in the empirical studies so far, the quality regulations has made it impossible to implement a fully agile software development practice. To obtain compatibility between agile and documentation-driven software development, an extension of either one or both methods is required. The review also showed that the main differences between agile and documentation-driven development are the way in which the requirements are handled and the amount of documentation expected/allowed. Handling these differences therefore become the main challenges when meshing. Both the agile and the documentation-driven methods are however flexible, so providing just enough documentation and treating these documents as deliverables at the iterations are the key guidelines. It is furthermore recommended to treat the documents as deliverables of the iterations. To satisfy the quality standards on the issue of requirements the agile user stories need to be translated into a business language and written in a requirements specification.

5.3 Adopting Quality Assurance Technology in Customer-Vendor Relationships: A Case Study of how Organizational Relationships Influence the Process

The purpose of this paper is to understand how the adoption of quality assurance can be influenced by the relationship between a customer and a software organisation. It is a study of the adoption of quality assurance in a small, agile software organisation. The software organisation consisted of 15 software developers, of which 8 developers were associated with the project of developing a safety-critical software device for drug dispensing for their main customer. The customer consisted of several organisations: the main customer organisation and 14 user organisations. The system consisted of a kernel version (for which the main customer was paying) and 14 local versions/adaptations (for which each user organisation was paying).

The software development practice was primarily agile in following way: The software was built in iterations, in close cooperation with the customer. Each month an administration meeting between the software organisation and the customer representatives took place. At these meetings the requirements for the kernel version of the device were discussed, determined and prioritised. The requirements for the local adaptations were accepted at all times primarily by email or phone. All requirements were written by or in cooperation with the customer and formed as user stories, specified in electronic forms. The knowledge sharing within the developer team was done primarily through face-to-face communication, the software developers were sitting close together and worked in pairs; and very little documentation was done. Testing was done continuously.

Due to the safety-criticality of the software, the customer demanded a certification according to the GAMP standard, which led to a struggle implementing documentation-driven elements in the software practice. Despite a pressure from the customer to adopt quality assurance, the software organisation was not able to achieve a mesh and pass an external audit. They failed mainly because sufficient documentation, of for example the requirements, was not provided.

It is a single, interpretive case study (Walsham 1995). The data collection and data analysis was conducted in 2 phases. The first phase focused on the software organisation and their challenges to adopt the quality assurance practices. As the study of the software organisation did not reveal the underlying cause as to why they were struggling with the adoption of the documentation-driven practices, the focus in the second phase was broadened to include the relationship between the software organisation and the customer. The study was based on 9 qualitative interviews with both developers and management from the software organisation and super users from both the main customer and the user organisations. Furthermore, observations on the software organisation and document-studies were conducted. The analysis of the influence of the interorganisational relationship on the adoption was done based on the framework of the attributes and processes of interorganisational relationships (Goles, Chin 2005).

The case study showed that it can be very difficult to adopt quality assurance in an agile software practice. The agile customer relations did influence the adoption. A successful trusting relationship between an IT vendor and a customer can be threatened by the adoption of the formal control mechanism quality assurance. The trusting and close relationship however also helped the parties overcome this threat and integrate some of the quality assurance practices into the development practice. Another main issue was the high amount of documentation and the strict approval process required by the quality standard.

5.4 Agile Software Development and its Compatibility with a Document-Driven Approach? A Case Study

The purpose of this paper is to provide an empirical understanding of how the agile and the documentation-driven methods can be meshed in practice, and how well this mesh is practiced. It is a study of a software group in a large pharmaceutical organisation, that was developing a safety-critical, embedded device for drug dispensing. By law, the device and therefore the software practice had to comply with the FDA standard. At the same time the software group had successfully implemented several practices from the agile method, Scrum.

The study was organised as an interpretive, single case study (Walsham 1995). The data was collected and analysed iteratively through three phases, including qualitative interviews with developers, testers and managers and a seminar presentation where the results were presented and discussed. The analysis was performed using Soft Systems Methodology (SMM) (Checkland, Scholes 1990): 1) rich pictures of the situation and problems encountered were drawn, 2) root definitions and conceptual models of activity systems were created, and 3) a comparison between the models and the interviews was conducted.

The case study showed that the software group was agile as they ran time-boxed iterations, their tasks were documented in product and sprint backlogs and they used a large Scrum board and burn down charts to display the progress of each iteration. The iterations were initiated with a sprint planning meeting as advocated by Scrum, but since the product was developed for a market, no immediate customer was associated and no customer did therefore participate in these meetings. The iterations were coordinated by daily Scrum meetings; both the sprint planning meetings and the daily Scrum meetings served as great opportunities for communication and knowledge sharing. At the same time, the software group used documentation-driven practices as the requirements were fixed early on in the project; they did not have any customer contact and the practice was affected by the amount of work products in terms of documentation.

This paper shows that it is possible to implement a software practice that uses elements of both agile and documentation-driven methods. This is however not easily achieved – the agile practices suffered from the adaptation to the FDA standard, the bureaucratic management of the project and the sequential work processes of the other project groups. Due to the changes to the hardware and user interface, the iterations were interrupted and sometimes even broken off to make time for the stabilisation of the code.

5.5 Introducing Agile Practices in a Documentation-Driven Software Development Process: A Case Study

This paper provides a detailed understanding of how a software group has implemented and improved a software practice meshing the agile and the documentation-driven methods. The purposes are to understand the possibility of meshing and the challenges to implementing a software development practice that meshes the agile and the documentation-driven software methods. The paper presents a longitudinal case study of a software practice used for developing a safety-critical, embedded device for drug-dispensing. The device was to be approved by the FDA and the software practice therefore needed to comply with the FDA standard, but at the same time the software group was improving their software development practice by adopting and adapting several Scrum practices. The paper describes the process of implementing this meshed software practice in the software group and the process of gradually improving the software practice and increasing the agility.

The paper is based on a longitudinal, interpretive single case study (Walsham 1995). The empirical data was collected and validated iteratively through two phases. The data consists of 26 qualitative interviews, observations, document studies and a seminar presentation of the findings. While the interviews in the first phase were organised as diagnostic interviews (Iversen et al. 1999); the interviews of the second phase were based on a framework of the home grounds of the agile and the documentation-driven methods. For the final analysis, the interviews were collected into one hermeneutic unit using ATLAS.ti V6 (Muhr 1991) and coded according to the framework of the characteristics of the home grounds of the agile and the documentation-driven methods (Boehm, Turner 2003a). The final analysis focused on the changes in the software practice from the first phase to the second phase, in order to determine the challenges of introducing the agile practices in the documentation-driven software practice. The full analysis was written up, presented in a report and validated by the software group.

The case study showed how the software group had implementing an iterative software development practice with several agile elements embedded in a documentation-driven project. The software practice was agile as they ran time-boxed iterations of 2 weeks, including a planning meeting and a retrospective. The developers found these short iterations advantageous; they were forced to break each task down, which made it easier to get an overview of the content of each task and thereby made it easier to estimate. The control of the tasks and the progress of the iterations were supported by the use of Scrum boards. To keep with the agile principle of small self-organising teams the developers were divided into two smaller teams. Within these teams, the communication and knowledge sharing was primarily personalized. The developers coordinated through daily stand-up meetings The Scrum Master facilitated the coordination between the two teams. The developers, however, found it very important to keep these meetings short in order to keep them from being too time-consuming and become uninteresting.

The software group and the software practice were, however, affected by the requirements of the FDA standard and the documentation-driven project. The project ran sequentially and controlled by long-term milestones. These milestones affected the activities of the software group; they were for example forced to write the documents in the early phases of the project as these were to be approved in order to pass a milestone. This made it difficult to handle the documents iteratively. The documents were placed in a hierarchy; the higher a document were placed, the stricter the approval process became and the harder it became changing these. Despite the personalized communication and knowledge sharing, the developers spent a lot of time writing documents in order to comply with the requirements of the FDA standard. The documents were only to a small extend used for knowledge sharing purposes as only a few developers actually read them; but, the documents were found useful for training of new employees. The requirements were fixed early in the project and as the product was developed for a market, there was no immediate customer associated; hence they had no customer contact or validation of the interpretation of the requirements before implementation. The requirements specifications were, however, rewritten in a late stage of the project. Many of the requirements were either not documented sufficiently or were documented in too much detail, several of the requirements were furthermore contradicting.

Implementing a meshed practice was not straightforward. The greatest challenge of meshing in this case study was implementing an agile test-driven and customer-driven practice. The project management was driven by the requirements of the FDA standard and did therefore not focus on testing in the early project stages, hence resources for implementing a test-driven development was not prioritised and allocated. Implementing a well-functioning product owner role was furthermore difficult due to lack of priority and resources from the project management. The late-testing and lack of customer relations were major disadvantages and the main weaknesses of the software practice. The analysis showed how the strengths of the software development practice were highly connected to the agile method. The iterative software development practice, the agile communication and knowledge sharing and the informative workspace were identified as the main strengths. But being able to comply with the quality standards and secure high quality and security of the software was, however, essential for the safety-critical product and therefore also considered a major strength.

5.6 Barriers to Adoption of Agile Software Development Practices: A Knowledge Transfer Framework

The focus of this paper is on the challenges of the adoption and diffusion of agile practices within an organisation. The research question was: “which barriers can prevent the transfer of knowledge of agile practices from one team to another?” The paper empirically addresses how knowledge and experience is transferred from one development team to another. The software group from which the knowledge had transferred has experimented with the implementation of agile practices in its software practice, while the receiving software group had very little experience in agile methods.

The study is based on a longitudinal and interpretive case study (Pettigrew 1990; Walsham 1995). The empirical data were collected over a period of one year (May 2010 to April 2011) in a large pharmaceutical organisation, with a focus on two software groups (A and B) and the context of which they were a part. Data were analysed and validated iteratively through two phases, each respectively focusing on software groups A and B. The data consisted of interviews, observations and document studies. The analysis of software group A mapped their practices and knowledge; the practices were compared to the agile and the documentation-driven practices. The analysis of software group B mapped the specific barriers hindering the knowledge transfer process. In order for this, the data was analysed based on a framework of barriers to the knowledge transfer of agile practices, that was developed from the literature on knowledge transfer and the literature on the adoption of agile practices.

Software group B had attempted to adopt the software development practice used by software group A, meshing the agile method Scrum with the requirements of the FDA standard. They aimed at implementing a software practice embedded in a documentation-driven project. Despite many initiatives software group B struggled to implement the agile practices in their software practice. For a short period they successfully implemented the iterations, but due to a shift in project focus, these were broken off. Software group B was however very determined to resume the iterations. The project was controlled by long-term milestones that determined the activities of the software group; because of the next milestone, the developers were for example focusing on determining the architecture of the system and writing the design document. The communication and knowledge sharing strategy in software group B was primarily personalized as they were only 3 developers. But, in order to comply with the FDA standard much documentation was also written. They were developing a product for a market and had no immediate customer associated. They were therefore struggling to implement a well-functioning product owner role; the role had been assigned to the coordinator of the project, but due to lack of priority at the project management he struggled to allocate the time needed to properly fill the product owner role. The project management did not prioritise software testing at this early stage in the project and no software testers had been associated. In total, the case study showed that several barriers hindered the knowledge transfer:

1) lack of time was a primary factor; the focus of the project was on developing prototypes and mechanical parts and not on the knowledge transfer and improvement of practice; time and resources were therefore not allocated by the project management,
2) the organisational culture of the project management and the software developers were not shared; the project management had very little experience with software development, instead they relied on the experience with the development of hardware and mechanical parts,
3) personal skills also played a role: just as the project management did not have experience in software development, none of the software developers had experience in using Scrum and needed to be trained in the method before applying it,
4) motivation and willingness was also a critical factor, as the project management did not shown much motivation and willingness to allocate time and resources to the adoption of the Scrum practices,
5) implementing the agile customer relation was proven difficult in this case, as the software group had problems implementing a well functioning product owner.

6. DISCUSSION

This thesis raises the research question: how can the agile and the documentation-driven methods be meshed in practice? This research question was pursued by two research efforts: a literature review, providing a theoretical foundation, and two case studies, providing empirical knowledge. In this chapter, the findings of the papers are revisited and discussed. Furthermore, the limitations of the research and the possibility for future research are discussed.

6.1 Meshing

In this section the definition of mesh is elaborated based on the existing literature. Figure 5 depicts the definition of meshing agile and documentation-driven software development. At each end of the line the pure form of respectively agile software development and documentation-driven software development is placed. Moving towards the centre of the line is still considered pure agile or pure documentation-driven as long as the paradigmatic differences are maintained. In the middle of the line a span of compatibility is placed. Compatibility is achieved by implementing both agile and documentation-driven elements in a software practice; the practice becomes meshed. The absolute mesh is a combination; a combination of the practice is achieved when elements of both the agile and the documentation-driven has been implemented are well-functioning and when neither agile nor documentation-driven elements are dominating. The point of combinable is placed in the middle of the line. Hence, being combinable also entails being compatible. On each side agile and documentation-driven software development are dominating, but not overshadowed by either. That is, it is possible to create a mesh which is primarily agile (this would be placed on the left side of the line within the bracket) or a mesh which is primarily documentation-driven (this would be placed on the right side of the line within the bracket).

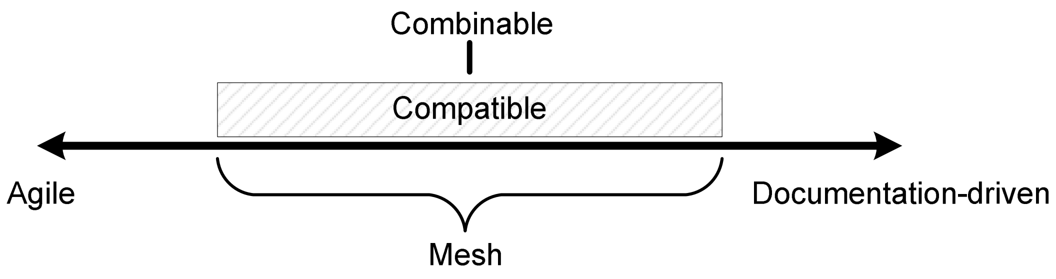


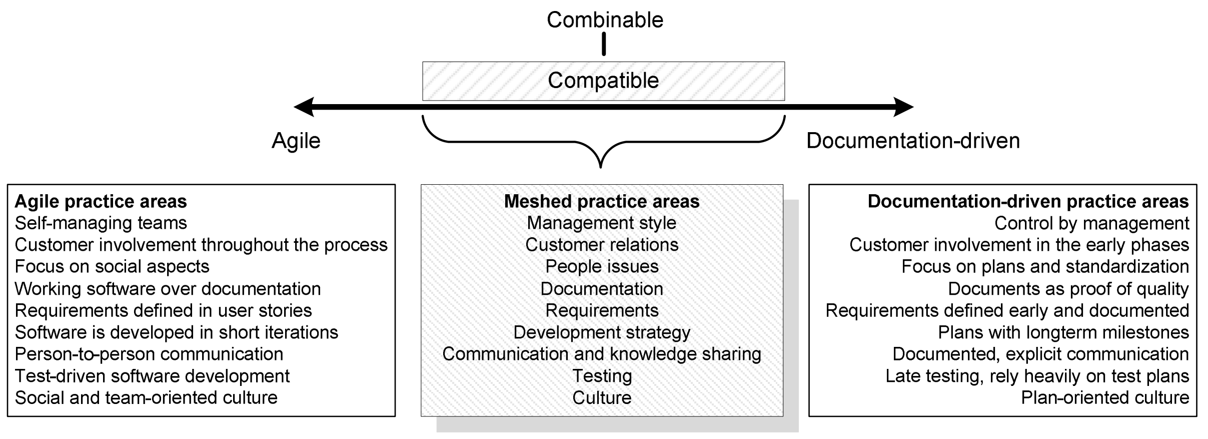
Figure 5. The mesh of agile and documentation-driven practice areas

In order to elaborate on this definition of meshing; the elements of agile and documentation-driven software development needs to be identified. These elements will identify the areas in which a practice is or potentially can be meshed. Based on the literature review nine such elements were identified (Table 6, Section 2.4). As this thesis is concerned with practice; in this discussion the nine elements will be treated as elements of practice; hence referred to as practice areas. Table 15 summarizes these practice areas for respectively agile and documentation-driven software development.

Practice area	Agile software development	Documentation-driven software development
Management strategy	Self-managing teams.	Control by management.
Customer relations	Customer involvement throughout the whole development.	Customer is involved during the early phase of the project.
People-issues	Focus on the social aspects.	Rely on documentation.
Documentation	Working software over documentation.	Use the documents as a proof of quality.
Requirements	Requirements are defined in user stories.	Requirements are documented.
Development Strategy	Iterative development strategy.	Sequential development strategy.
Communications and knowledge sharing	Person-to-person communication (personalization).	Documented, explicit knowledge (codification).
Testing	Test-driven software development.	Late testing and rely on test plans.
Culture	Social and team-oriented.	Plan-oriented.

Table 15.The agile and documentation-driven practice areas (cf. table 6, section 2.4)

Based on the practice areas, figure 6 depicts an elaborated version of figure 5. Pure agile and pure documentation-driven software development (placed at the ends) are characterised by their respective practice areas (represented in the left and right boxes). A practice area become meshed (represented in the middle box) when compatibility and maybe even combinability is achieved. The case studies showed that a software practice can be meshed by meshing each practice area. Each of these may however have different meshes; that is, the mesh of one practice area may be primarily agile while a mesh on a second practice area may be primarily documentation-driven. Even though some of the practice areas are not meshed an overall mesh of the software practice can still be achieved. Using these definitions, the mesh of the overall practice and the mesh of each practice area in the two case studies will be discussed in more detail below.



Agile practice areas
Self-managing teams
Customer involvement throughout the process
Focus on social aspects
Working software over documentation
Requirements defined in user stories
Software is developed in short iterations
Person-to-person communication
Test-driven software development
Social and team-oriented culture

Meshed practice areas
Management style
Customer relations
People issues
Documentation
Requirements
Development strategy
Communication and knowledge sharing
Testing
Culture

Documentation-driven practice areas
Control by management
Customer involvement in the early phases
Focus on plans and standardization
Documents as proof of quality
Requirements defined early and documented
Plans with longterm milestones
Documented, explicit communication
Late testing, rely heavily on test plans
Plan-oriented culture

Figure 6. The mesh of agile and documentation-driven practice areas

6.2 The Overall Mesh in the Case Studies

This section describes the overall mesh in the case studies. The declared goal of the software developers and managers, in the two case studies was to achieve a mesh. Figure 7, gives an overview of the overall mesh in the cases at the end of the studies.

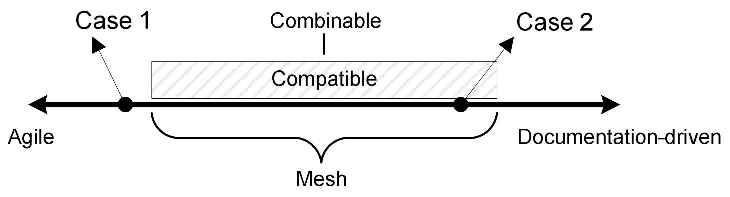


Figure 7. The overall mesh in the two case studies

Meshing in Case 1

Despite their efforts to achieve a mesh the development practice in the first case study remained primarily agile. Hence, on the line on meshing, the first case study is placed on the left side outside of the mesh bracket (see figure 7). The external audit showed that the excessive documentation required by the quality standard was the main reason why the mesh did not succeed; the amount and quality of documentation produced on requirements and for knowledge sharing purposes was not enough. Frequent customer contact was maintained at monthly administration meetings and by email or phone. The communication and knowledge sharing between the developers was primarily based on personalization, they worked close together and sometimes in pairs. The analysis of the customer relations showed a great trust between the parties. Due to this trust neither the developers nor the customers felt a need for a more documentation-driven customer relation and or a need for more documentation.

Meshing in Case 2

During the time of study the two software groups increased the agility of the software practices and achieved a mesh; the mesh was however primarily documentation-driven. Hence, case study two is on the line of meshing placed on the right side inside the bracket (see figure 7). Both software groups were successful at implementing an agile software development practice embedded in the documentation-driven projects. The analysis showed how the products were developed for a market and how the software developers were not able to establish a contact to a customer and implement a well-functioning product owner role. Software group B did however create a mesh on the practice area of customer relation by assigning the product owner role to the project coordinator, but not without problems (expanded below). Due to the excessive amount of documentation required by the FDA standard the practice area on documentation was very difficult to mesh. The case study also showed that handling the requirements in an iterative manner was difficult. This challenge arose due to a sequential treatment of the requirements specifications (forced by the sequential development strategy of the overall project). Software group A and partly also software group B meshed the practice area on development strategy by embedding an iterative software practice in the sequential project. The communication and knowledge sharing strategies of both software groups were in practice primarily personalized but also codified as much knowledge was documented. The testing in software group A was done in the late stages of the project and was based on thorough test plans, they were therefore primarily documentation-driven in the practice area of testing. Learning from software group A's experiences, software group B, however, had a great focus on creating a mesh in this area by introducing a more test-driven software development.

Interconnection of the Practice Areas

The analysis of the case studies reveals how the practice areas are highly connected and dependent of each other. The practice areas of 'requirements' and 'communication and knowledge sharing strategy' depend on the practice area of ’documentation’. The main reason for the challenges meshing these practice areas are connected to the fact that excessive documentation is required by the quality standards, hence the requirements also need to be documented in details and the knowledge sharing (even though a primarily personalized strategy is preferred) is forced to rely heavily on codification. The documentation is dependent on the development strategy; in case study two, the software developers found it difficult to handle the documentation in an iterative manner. The sequential development strategy of the project and the milestones determined when each document needed approval. How each practice area was meshed and the difficulties of meshing these are described in more detail in the following subsections.

6.3 The Mesh of each Practice Area in the Case Studies

The two case studies made it possible to provide contributions on how to create a mesh on six of the nine practice areas: 1) customer relations, 2) documentation, 3) requirements, 4) development strategy, 5) communication and knowledge sharing and 6) testing. This section discusses how the six practice areas were meshed in practice. Only a few guidelines are given in the previous literature on how to mesh each of the practice areas (section 2.5), instead the literature either deals with the pure forms or the possibility of creating an overall mesh. Each subsection does, however, discuss the contributions according to the few recommendations given in the literature.

6.3.1 Customer Relations

The agile practice area on customer relations recommends close customer contact throughout the development lifecycle to ensure that the end-product meets the needs and expectations of the customer. The documentation-driven practice area only involves the customer at the start of the project when creating the requirements (Boehm, Turner 2003c). This poses some challenges when trying to mesh customer relations. Figure 8 gives an overview of the mesh in the case studies on the practice area of customer relations.

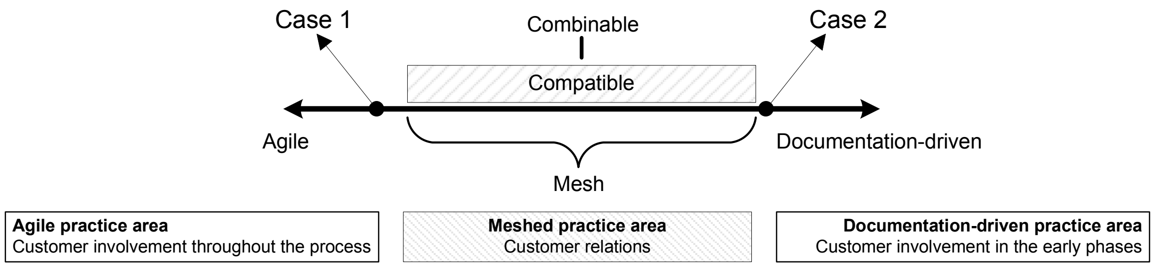


Figure 8. The mesh of customer relations in the case studies

Meshing in Case 1

The analysis of case study 1 showed how the software organisation and the customer had a close cooperation; every month a steering committee meeting with the representative from the central department and representatives from each user site took place. At these meeting the requirements for the kernel version of the system were discussed and determined. Additionally frequent contact through email and phone was maintained. This served as the way the requirements for the local versions were discussed and determined (Section 5.3, paper 2).

For customer relations the challenges of meshing in case study 1 is highly related to documentation. As a result of the agile customer relations a high level of trust has been built between the software organisation and its customer. As both parties found this close relationship valuable they had no wish to change it and include more documentation to prove the quality of the system. This shows that agile customer relations may have an impact on the implementation of documentation-driven practice areas. The trusting relationship between the software organisation and the customer was to some degree threatened by this. The trusting and agile customer relations did however also help the parties overcome conflicts brought on by some of the requirements of the GAMP standard (Section 5.3, paper 2). Hence, the customer relation was not meshed and remained primarily agile (see figure 8).

Meshing in Case 2

The analysis of case study 2 showed how both software group A and B were developing products for a market and had no immediate customer associated. Instead the requirements were based on the analysis done by the marketing group. Software group A therefore tried to get a representative from the marketing group to fill the product owner role. As they did not succeed in this, an internal person was assigned this role (Section 5.5, paper 4). Based on the experiences of software group A, software group B focused on having the product owner role filled by an external person early in the project. They succeeded and the role was assigned to the overall project coordinator, who had an overview and responsibility of all three fields: software, hardware and mechanics (Section 5.6, paper 5).

For customer relations the main challenge in case study 2 was implementing a well-functioning product owner role. The challenge arose as convincing the management of the importance of this role was difficult. For software group A the difficulties arose as the marketing group did not have the necessary resources allocated. The project management did not recognise the need for more agility in the overall project and resources were therefore not allocated (Section 5.5, paper 4). Software group B was successful in involving an external person, but difficulties arose as sufficient resources had not been allocated and he was hindered by other tasks and therefore not fully able to fulfil the role of the product owner (Section 5.6, paper 5). Hence, the practice area of customer relations of software group A was not meshed successfully and the mesh created in software group B was not well-functioning and needed improvement. Case study 2 is therefore represented right outside the mesh bracket (see figure 8). This highlights the importance of convincing the management of the importance of the product owner role. To do this, a realization of the advantages of the agile software development is essential.

Contributions in Relation to the Literature

So far, the literature on agile software development and on meshing does not focus on how to implement a well-functioning customer relation. The literature does acknowledge that the outcome of a project is influenced by the customer relations (Das, Teng 2001; Goles, Chin 2005). No research focuses on the influence of the customer relations on the adoption of documentation-driven elements in an agile software organisation. The first case study showed that close customer relations have great influence on meshing; it can both advance and hinder this. When trying to create a mesh it is therefore very important to be aware of how agile customer relations influence the implementation of documentation-driven practice areas.

Customer involvement is much heavier in the agile practice area (Huo et al. 2004) and it is also acknowledged in the literature that maintaining such a close customer relation requires a lot of time and involvement. Agile customer relations are therefore hard to implement (Paulk 2001; Paulk 2002). Case study 2 showed how both software group A and B were struggling to implement the product owner role of Scrum. They were developing products for a market and therefore they did not have an immediate customer associated. To solve this issue Murru et al (2003) propose filling this role completely by an internal person or shared between an internal and an external person. The second case study, however, showed that filling the product owner role by an internal person is very difficult as well, adequate time needs to be allocated the person assigned the product owner role. This requires the management to acknowledge the importance of this role.

Conclusion

Case study 1 showed that implementing documentation-driven customer relations may be hindered by an agile, trusting customer relation. For agile software organizations attempting to mesh, it is important to be aware of how agile customer relations impact the implementation of documentation-driven practice areas. Case study 2 showed that implementing a well-functioning product-owner role in a documentation-driven project was difficult because the management did not prioritize the resources needed for a product owner. To convince the management of the importance of a product owner role, diffusion of knowledge on agile customer relations is important. Based on this, it can be concluded that the customer relations of the agile and the documentation-driven practice area is difficult to mesh.

> **Proposition 1:** the practice area of customer relations is difficult to mesh. Implementing documentation-driven customer relations in an agile practice is hard as the level of trust can hinder this. When implementing an agile product owner role in a documentation-driven project highlighting the importance of this role to the management is essential.

6.3.2 Documentation

Documentation-driven software development relies on documentation to prove the quality of the software (Section 2.4, Paetsch et al. 2003). Agile software development focuses on working software over documentation (Section 2.4, Beck et al. 2001). Meshing the practice area of documentation therefore poses several difficulties. In this thesis meshing the practice area of documentation was proven the most difficult. Figure 9 gives an overview of the mesh in the case studies on the practice area of documentation.

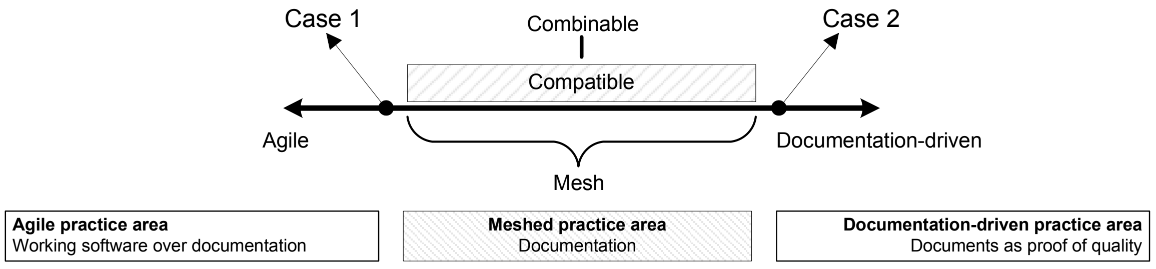


Figure 9. The mesh of the practice area of documentation in the case studies

Meshing in Case1

The analysis of the practice area of documentation in case study 1 showed how the developers of the small software organization did produce some documentation. The document study (section 4.3 and 5.3) showed how the requirements were documented by using a standard form identifying the customer (local user site or central department), the developer who specified the requirement and a short description of the requirement concerned. Most of the other documentation was written in the code. For larger tasks a design document were created. The developers used an electronic system to store and handle the changes of the documents (Section 5.3, paper 2).

The software organization did try to achieve a mesh, but was not able to meet the requirements of the GAMP standard, mainly due to the amount of documentation required by the standard (Section 5.3, paper 2). The challenges of meshing arose due to the strict way the GAMP standard required the documents to be handled. Besides being required to document, for example, the code and the reviews of the code; the software organization was required to handle the documentation in a strict manner keeping track of each change in the document and who had changed it. The electronic system they used for handling and storing the documents could not be approved by the GAMP standard; a certification of each tool used in the development practice was required. Hence, case study 1 is on the line of meshing placed on the left side, outside the mesh bracket (see figure 9).

Meshing in Case 2

The analysis of case study 2 showed how both software groups (A and B) spent much of their time writing documents. They documented for example: the design, the code, the reviews of the code and as the software was safety-critical, the possible safety-hazards and the strategies for handling these (Sections 5.4, 5.5 and 5.6, papers 3, 4 and 5). Many of the documents were written in the beginning of the project (for example the requirements specification and the design document). Getting the documents approved was essential for passing the early milestones of the project. Each document underwent a strict approval process, which made it hard to change the documents later in the project. The higher a document was placed in the hierarchy of documents, the harder it was to change it because of the approval procedure (Section 5.5, paper 4).

The analysis showed that for documentation, the greatest challenge of meshing in case study 2 was handling the documents in a light and iterative manner. The software groups attempted to handle the documents iteratively by writing these in increments, improving these as new information or experience came up and by treating these as output of the sprints (Section 5.5, paper 4). In order to pass the early milestones of the project, several of the documents had to be approved; this forced the developers to finish the documents in the early stages of the project. The bureaucratic and time-consuming process of getting a document approved or changed was therefore the main challenge of implementing an iterative strategy for handling the documents. The communication in software group A was mainly agile and personalised, the purpose of the documentation was therefore primarily to satisfy the requirements of the FDA standard (Section 5.5, paper 4). Hence, case study 2 is on the line of meshing placed on the right side, outside the meshing bracket (see figure 9).

Contributions in Relation to the Literature

The literature review identified the amount of documentation required by a quality standard as being very different from the agile focus on working software over excessive documentation (Section 5.2, paper 1). The literature underlines that, it is not necessary to write a large amount of documentation to comply with quality standards (Bos, Vriens 2004). Furthermore, the agile principle of working software need not be a rejection on writing documentation. But, agile methods do still not support the degree of documentation demanded by documentation-driven methods (Paetsch et al. 2003). Documentation-driven methods can result in extensive documentation, and in these cases a lot of time will be spent on writing documents instead of developing software (Paetsch et al. 2003). Kähkönen and Abrahamsson (2004) report on a case study in which an organization had implemented a light documentation practice in compliance with XP, but to satisfy CMMI they had to produce additional documentation. Both case studies of this thesis show specifically that meshing the agile and documentation-driven way of handling the documentation was difficult. The developers in both case studies spent much time writing documents and found it, due to the strict quality standards, difficult to keep an agile focus on working software over documentation. McMichael and Lombardi ((2007)) underline that most standards are flexible and that one needs to find the appropriate level of documentation. This may be harder for safety-critical projects as the quality standards are stricter in order to make sure quality is documented sufficiently (Wright 2003). Another recommendation in the literature is to treat the written documents as deliverables at the end of an iteration (Namioka, Bran 2004; Stålhane, Hanssen 2008). The second case study showed that this can be very difficult in an agile software practice embedded in a documentation-driven project, because approval of the documents will be required in the early project phases in order to pass the first milestones.

Conclusion

The practice area of documentation is the hardest to mesh. Complying with a quality standard (for safety-critical software) requires substantial documentation and a strict manner of handling these. Even though the agile practice area on documentation is flexible, it is hard to keep with the principle of a focus on working software over comprehensive documentation. Case study 2 also showed that handling the documentation in an iterative manner may be difficult due to the milestones proposed by the documentation-driven development strategy. Due to the strict requirements of documentation provided by the quality standards, a mesh on documentation (at least in the cases of safety-critical software development) will most likely be primarily documentation-driven; hence placed on the right side of the line.

> **Proposition 2:** the practice area of documentation is the hardest to mesh. The amount of documentation required by quality standards is not supported by the agile focus on working software. Handling the documents in a light and iterative manner may be difficult in a project controlled by long-term milestones and approval of the documents in the early stages.

6.3.3 Requirements

The agile practice area on requirements advocates that the requirements are gathered throughout the iterative lifecycle (Section 2.4., Cockburn 2006). In documentation-driven development (such as the Waterfall model), the requirements are gathered and specified at the start of the project (Section 2.4, Pressman 2000). Due to the documentation-driven focus on documentation of the requirements this practice area is also difficult to mesh. Figure 10 gives an overview of the mesh in the case studies on the practice area of requirements.

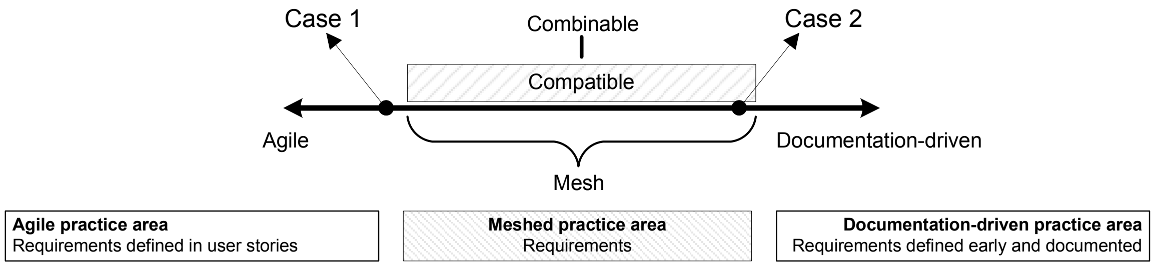


Figure 10. The mesh of the practice area of requirements in the case studies

Meshing in Case1

The analysis of the practice area of requirements in the first case study showed that the small agile software organization gathered its requirements by several means; at the monthly administration meetings and by email and phone. The requirements for the kernel version were gathered at the administration meetings. At these meetings a representative for the central department and representatives for each user sites were present. Each suggestion for improvements was discussed and the requirements agreed on were prioritized. The requirements were documented in a specification. The requirements for the local adaptations were primarily gathered by phone and email and specified by the customer in a specific form (Section 5.3, paper 2).

The analysis showed how the small software organization though they tried did not achieve a mesh in this area either. For requirements, the challenges of meshing in case 1 arose as, the way the requirements were handled could not be determined as a mesh. The documentation provided in terms of electronic forms could not be approved by the GAMP standard as proper documentation of the requirements. Due to a close, trusting relationship and due to a sufficient level of quality on the current version of the system, neither the software developers nor the customer perceived a need for further documentation of the requirements. The customers did not seek additional proof of quality. The close relationship and frequent contact between the customers and developers eased the possibility of getting more detailed information on a certain requirement, if needed. Hence, on the line of meshing case study 1 is on the practice area of requirements placed on the left side, outside the mesh bracket (see figure 10).

Meshing in Case 2

The analysis of case study 2 showed how the requirements were gathered by the marketing group at the start of the projects; this applied for both software group A and software group B. All requirements were specified in several requirements specifications. The marketing group was in charge of the documents on the high level of the hierarchy, while the software groups were in charge of making the detailed design of each requirement and document these in the software requirement specification (Section 5.4, paper 3).

Even though the requirements were based on a thorough analysis, some requirements (in the specifications) were contradicting, some were over specified and other requirements were not specified enough. A decision to rewrite the requirements specifications was therefore taken later in the project; this was very time consuming (Section 5.5, paper 4). Rewriting the requirements made them much easier to understand and implement. This shows that redefinitions of the requirements can be both necessary and advantageous. Even though the documentation-driven methods rely on a thorough analysis before defining the requirements, these may still change and a rewriting of the specifications might be required. Based on this analysis, the second case study is therefore (on the line of meshing) placed on the right side of the line, just inside the mesh bracket (see figure 10).

Contributions in Relation to the Literature

The literature review identified the way the requirements are handled as a major challenge to meshing the agile and the documentation-driven methods (Section 5.2, paper 1). Boehm and Turner (2005) highlight that the differences on handling requirements proposed by the agile and the documentation-driven practice areas can cause problems when attempting to mesh. Both case studies of this thesis specifically showed that meshing the practice area of requirements was difficult due to several issues. User stories written in a plain business-like language cannot be used directly as requirements by an organization wishing to comply with a quality standard (Wright 2003). Instead, each user story needs to be translated into functional test cases (Beznosov 2003). The first case study showed that the requirements written by the customers could not be approved by the GAMP standard. Furthermore, it showed that due to a close, trusting customer relation the need for further documentation of the requirements was not perceived. This was the main reason why they were not able to achieve a mesh.

Paetsch et al. (2003) state that the agile and the documentation-driven practice areas on requirements are pursuing similar goals, but the major difference is the emphasis on the amount of documentation needed. The case studies of this thesis support this. Case study 1 showed that, the small software organization handling the requirements in an agile manner did not comply with the requirements of documentation. The second case study also showed that the documentation of the requirements was hard to mesh. The requirements were gathered through a big upfront analysis, but software group A did however feel a need for an iterative requirements process and a mesh was therefore attempted; hence they changed the requirements and documents in the late stages. Doing so is expensive and a more iterative requirements gathering would therefore be preferable (Beck, Andres 2004). To achieve a mesh, the literature proposes introducing a requirements engineering phase in the beginning, but still adjusting requirements iteratively (Namioka, Bran 2004; Nawrocki et al. 2002). Handling the requirements specifications in an iterative manner was in case study 2, however, difficult due to the reasons highlighted in the section on the practice area of documentation. These challenges most likely arose due to the way a mesh was created in the second case study (an agile software development embedded in a documentation-driven project).

Conclusion

Meshing the practice area of requirements is difficult. The main challenges are closely linked to the practice area of documentation. In general it is difficult to handle the documentation in an iterative manner due to a strict approval process of documents, this issue also applies to the requirements specifications. In order to comply with a quality standard, requirements furthermore need to be specified more rigidly than the agile user cases propose.

> **Proposition 3:** the practice area of requirements is difficult to mesh. The main challenge is the different emphasis on the amount of documentation needed and the difficulties of handling the requirements specifications iteratively.

6.3.4 Development Strategy

The practice areas on development strategies also differ, as the agile practice area suggests an iterative software development while the documentation-driven practice area relies on long-term plans and milestones (Section 2.4, Larman 2004). The developers of case study 1 did not attempt to mesh the practice area on development strategy; figure 11 therefore only gives an overview of the mesh in case study 2 on this practice area.

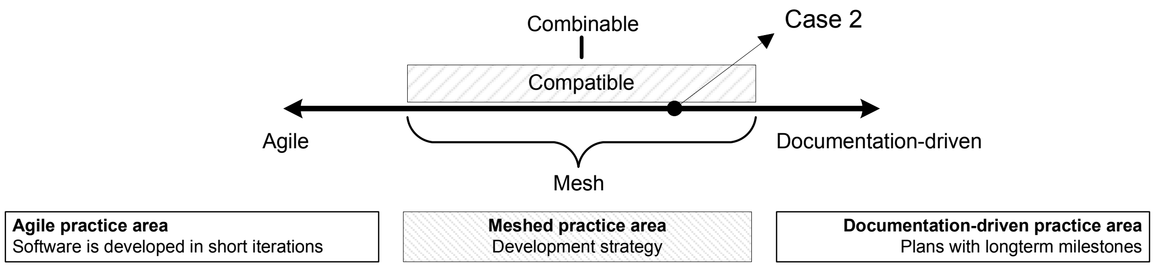


Figure 11. The mesh of the practice area of development strategy in case study 2

Meshing in Case 2

The analysis of the second case study showed how software group A found it possible and advantageous to implement the software in iterations and increments, even though the project was controlled by an overall plan divided into several long-term milestones (Section 5.5, paper 4). The sprints were coordinated through planning meetings at the beginning of each sprint and through a daily Scrum meeting. The tasks were controlled by the use of a product backlog and sprint backlogs (section 5.4, paper 3). At the beginning, software group A sometimes broke the iterations off because of instability of the software and because of urgent tasks coming in from other project groups working sequentially (Section 5.4, paper 3). Software group B had also partly managed to implement iterations of two weeks length in their software practice (Section 5.6, paper 5). Software group A experimented with the length of their iterations; in the beginning they ran sprints of 30 days length, as suggested by Scrum (Section 2.2, Schwaber, Beedle 2001), through time decreased the sprint length to two weeks. They found that short iterations force the developers to divide the tasks into smaller parts in order to be able to solve the task during a sprint. Dividing the tasks makes it easier to get an overview of the content and therefore easier to estimate (Section 5.5, paper 4).

The analysis however also revealed some challenges. The agile, iterative software development practice was affected by the long-term milestones of the documentation-driven project. For example, to pass one milestone the design specification had to be approved. This lowered the influence of both software group A and software group B on the content of the iterations and (as discussed in the section on the practice area of documentation) hard to handle the documentation in an iterative manner. Another challenge arose due to the lack of customer relations; this made it difficult to run the iterations according to the agile practice area. According to Scrum (Section 2.2, Schwaber, Beedle 2001) each sprint is initiated with a planning meeting and a sprint review meeting. At these meetings the customer is included and the new functionality of the increment is demonstrated. Software group A did initiate the sprint with a planning meeting and a sprint review meeting, but since no customer was associated, this part of the meeting did not fulfil the recommendations of Scrum (Section 5.4 and 5.5, papers 3 and 4). Software group B struggled to include their product owner in the planning meetings (Section 5.6, paper 5). This thesis therefore showed that it is possible to implement a highly agile and iterative software practice embedded within long-term milestones and such a practice has several strengths, but also challenges. Hence, on the line of meshing case study 2 is placed on the right side, well inside the mesh bracket (see figure 11).

Contributions in Relation to the Literature

Previous literature proposed that the development strategies of the agile and the documentation-driven methods were incompatible (Section 2.4, Nerur et al. 2005). The analysis of software group A showed that it is possible to mesh the development strategies by embedding an iterative agile software practice in a documentation-driven project controlled by milestones. The milestones did, however, affect the content of the iterations and forced the documents to be approved in the early stages of the project. Boehm and Turner (2005) advocate that the milestones are realigned and redefined to better fit an iterative approach. An empirical study of how to do this was however not presented. The literature acknowledges the advantages of iterative software development (Rising, Janoff 2002). In their paper Rising and Janoff (2002) highlights the advantage of developing the software in iterations in order to be able to define the requirements up front. Their study also points to the fact that the developers are forced to break the tasks down into manageable parts as a main advantage of the iterations.

Conclusion

The agile and the documentation-driven practice areas on development strategy can be meshed. The case study showed that it is possible to implement a practice in which a highly iterative software development is embedded in a sequential project strategy controlled by long-term milestones. The milestones advocated by the documentation-driven practice area may however affect the agile iterations, if they cannot be realigned, achieving combination may be impossible. It is however still possible to gain several advantages of the iterative development strategy.

> **Proposition 4:** the practice area of development strategy can be meshed. One way is to embed an agile, interview software development strategy in a documentation-driven project. Such a practice has several advantages (such as being forced to break the tasks down) but it does pose some challenges, as the long-term milestones will affect the content of the iterations.

6.3.5 Communication and Knowledge Sharing Strategies

While the agile practice area on communication and knowledge sharing advocates a personalisation strategy, the documentation-driven practice area advocates a codification strategy, in which knowledge is documented (Section 2.4, Boehm, Turner 2003c; Crawford et al. 2006). Figure 12 gives an overview of the mesh in the case studies on the practice area of communication and knowledge sharing strategy.

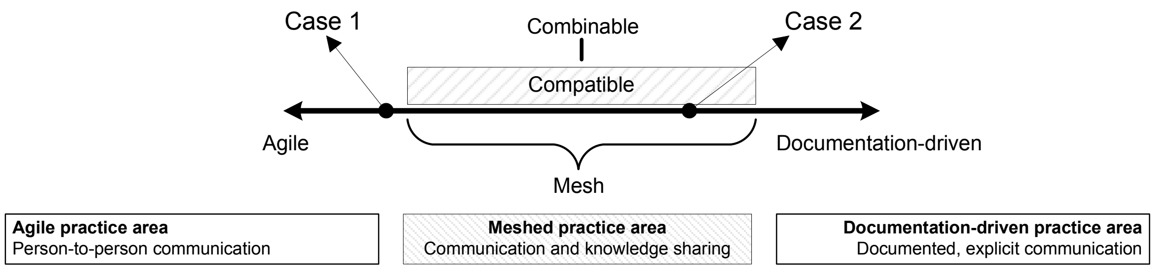

Case 1
Combinable
Case 2
Compatible
Agile
Documentation-driven
Mesh
Agile practice area
Person-to-person communication
Meshed practice area
Communication and knowledge sharing
Documentation-driven practice area
Documented, explicit communication

Figure 12. The mesh of the practice area of communication and knowledge sharing strategy in the case studies

Meshing in Case 1

The analysis of the first case study showed that the small software organization primarily based their communication and knowledge sharing on a personalization strategy. The communication and knowledge sharing between the developers were facilitated by the software developers sitting close together (Section 5.3, paper 2). The developers had frequent small meetings and they worked in pairs; the latter especially in order to pass knowledge on from experienced developers to less experienced developers. They only documented a smaller amount of the knowledge.

The analysis of the challenges showed that, one reason why the software organization was not able to pass the GAMP certification was the lack of documentation. The developers were however not able to increase the amount of documentation and create a mesh on this practice area. They perceived the personalization strategy most useful in practice and did not perceive additional documentation to be necessary. Hence, on the line of meshing case study 1 is placed on the left side, outside the mesh bracket (see figure 12).

Meshing in Case 2

The analysis of case study 2 showed that software group A in practice primarily used a personalised communication and knowledge sharing strategy. The daily Scrum meetings and the planning meetings at the start of each iteration served as great opportunities for communicating and sharing knowledge in the development teams (Sections 5.4 and 5.5, papers 3 and 4). Software group A meshed the agile and the documentation-driven communication and knowledge sharing strategies as they also codified their knowledge in documents. They did this mainly because it was required by the FDA standard. To support communication and knowledge sharing, software group A used several information radiators, for example two Scrum boards (one for each developer team) displaying the status and progress of the iteration. These Scrum boards served as a great way of getting important information instantly with minimal effort and they supported the daily Scrum meetings, making the tasks visible. Information radiators are an agile strategy to pass along information quietly, with little effort, and without disturbing people (Cockburn 2006). Software group B tried to implement a similar iterative Scrum practice with stand up meetings, retrospectives and planning meetings, but only succeed partially (Section 5.6, paper 5). Their communication strategy was however also mainly personalization as they were only three persons working close together.

The analysis of the challenges showed that even though the software developers of software group A created a mesh which was primarily documentation-driven, they did find the agile personalisation strategy the most useful in practice. Only little knowledge was actually shared by the many documents that were written, as only few people actually read them. The developers however found the documentation important for the training of new people (Section 5.5, paper 4). The developers however, also found it very important to keep the Scrum meetings as short as possible, as otherwise it would take up too much time and become boring (Section 5.5, paper 4). In the interest of keeping the daily Scrum meetings short, the developers held them standing up (Kniberg 2007). Hence, on the line of meshing case study 2 is placed on the right side, inside the mesh bracket (see figure 12).

Contributions in Relation to the Literature

There is no literature on meshing the agile and the documentation-driven practices that deals with the issue of meshing the practice area on communication and knowledge sharing strategies. The literature on knowledge management however reports on meshing communication strategies. In line with case study 2 of this thesis, Hansen et al. (1999) concludes that an organization needs a knowledge sharing strategy that consists of both personalisation and codification. He advocates a distribution of 80%-20%. These finding have been supported by for example Kautz and Thaysen (2001). The two case studies of this thesis found it advantageous to have a primarily personalized communication strategy, but due to the quality standards forced to have/introduce a dominating codification strategy.

Conclusion

The agile and the documentation-driven practice areas can also be meshed, but a combination has not been proven. It is possible and advocated to create a communication strategy which meshes personalisation and codification. The extensive focus on codification required by the quality standards (especially in safety-critical development) will however most likely dominate the agile personalization strategy.

> **Proposition 5:** the practice area of communication and knowledge sharing can be meshed, but combinability has not been determined. Creating a mesh between the agile personalization strategy and the documentation-driven personalization strategy is advocated. A mesh will most likely be dominated by the codification strategy, due to the requirements of the quality standards.

6.3.6 Testing

Testing is crucial in agile development and test-driven development has become increasingly popular within the agile community (Nerur et al. 2005). It is not only suggested that test cases are written before the software, it is also most importantly suggested that there is testing in of all parts of an increment and that completion is determined by the increment passing all tests (Beck, Andres 2004). The documentation-driven practice area suggests that testing is done in the final stage of the project and that tests rely on detailed test plans (Boehm, Turner 2003c). The developers of case study 1 did not attempt to mesh their testing strategy; figure 13 therefore only gives an overview of the mesh in case study 2 on the practice area of testing.

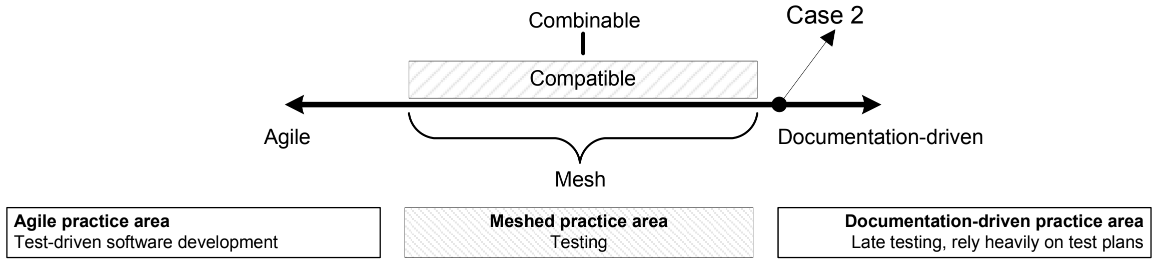

Combinable
Case 2
Compatible
Agile
Documentation-driven
Mesh
Agile practice area
Test-driven software development
Meshed practice area
Testing
Documentation-driven practice area
Late testing, rely heavily on test plans

Figure 13. The mesh of the practice area of testing in case study 2

Meshing in Case 2

The analysis of case study 2 showed how software group A wanted to develop the software through time-boxed iterations with a fully implemented and tested increment as output, but did not succeed. The increments were often not fully tested before they were moved on for system test. Much testing was postponed until the later stages of the project, and to comply with the quality standards much effort was put on creating test plans and procedures. Many of the developers did not see any value in unit testing, and the implementation of new functionality seemed to be valued much higher than testing. In general, in the project, testing of the software was not a focus (Section 5.5, paper 4). Trying to avoid the same problems as software group A, software group B had a focus on implementing a test-driven software practice. They were however facing problems convincing the project management that early testing is important. The project did therefore not yet, include any software testers at the time of study (Section 5.6, paper 5). The lack of testing and late testing was a serious issue, and perhaps the main weakness of the software practice of software group A. The lack of continuous testing of the software led to severe problems in keeping to the continuous integration of the software, and left software group A with a large amount of errors in the software, on which they had to spent a lot of time correcting (Section 5.4 and 5.5, papers 3 and 4). Errors detected at the system tests had to be handled much more formally than errors detected at the unit tests and integration tests, which decreased the agility.

The analysis showed that implementing a test-driven practice in a documentation-driven project, may not be straight-forward. For testing, the main challenge of meshing was convincing the management of the importance of test-driven software development. In a documentation-driven project the management will most likely be driven by the requirements of quality standards, that do not focus on tests in the early stages of the project. Hence, on the line of meshing, case study 2 is placed in the right side, outside the mesh bracket (see figure 13). Case study 2 showed that resolving the issue was not easy. Increasing and improving the unit tests and implementing a test-driven practice would have required a significant change in the mindset of the developers. It would also require the project management to realise the need for a focus on testing in the early phases of the project. Education is one way to increase the focus on test-driven development, both among the developers and the management.

Contributions in Relation to the Literature

Boehm and Turner (2005) suggest that implementing agile practices such as test-driven development is a way to mesh the agile and the documentation-driven methods. They argue that documentation-driven methods already focus on test and quality of the software, while the test-first and continuous integration proposed by the agile methods will be helpful in finding the problems earlier rather than later. Jakobsen and Johnson (2008) report on a case study in which Scrum and CMMI level 5 have been meshed; they underline how the test practices of Scrum (test-driven development and automated tests) supported the iterations and were an important part of the quality assurance. Explicit quality plans were used to ensure testing was done in the right manner. The literature does not propose that implementing agile test-driven development may be difficult due to the different focus in documentation-driven projects in the implementation phase, as case study 2 showed.

Conclusion

The strategies for testing advocated by the agile and the documentation-driven practice areas are also difficult to mesh; it can be difficult to implement a test-driven practice in an organization driven by the requirements of a quality standard, as testing is not prioritised until late in the project. In order to create a mesh in this area it is important to diffuse the knowledge of agile test-driven development to the entire project and especially to the management.

> **Proposition 6:** the practice area of testing is difficult to mesh. Implementing an agile, test-driven practice requires that both developers and the management realize the advantages of test-driven development. In a documentation-driven project education on agile testing is useful.

6.4 Limitations

This section discusses the limitations of this thesis in terms of the research method and the cases chosen to answer the research question.

The Research Method

Case study research allows the researcher to study a phenomenon in a natural setting, to learn about the state of the art and to generate theories from practice (Benbasat et al. 1987). However, like all other research methods case study research has some general limitations, that also apply in this study. Even though it is not possible to remove these limitations completely, some efforts have been done in order to minimize these.

One concern of case study research has been the lesser rigor (Yin 2009). The collection and the analysis of data are dependent on the integrative powers of the researcher (Benbasat et al. 1987). Providing a systematic approach to data collection and analysis is one way of ensuring rigor. Validation of the analysis and of the findings is another. Working in close cooperation with other researchers is a third. To reduce the limitations of rigor all three have been applied in this thesis. The research approach (Chapter 4) describes the systematic approach to the data collection and analysis. Both the overall research design and the research approach used for conducting the literature review and in each case study have been carefully planned and executed with a basis in the current theories. The findings of the case studies were validated iteratively, both by conducting seminars and written reports. This way the software developers and the management were included in the validation of the analyses. Three of the five research papers have been written together with my co-workers providing different views on the data and the analyses.

Another limitation is that even though the researcher is an outside observer, he or she will in some sense always influence what is happening (Walsham 1995), for example may a tape-recording make the interviewee less open or less truthful (Walsham 2006). To minimize this limitation, iterative, long-term in-depth case studies were conducted. By including the same interviewees more than once, they had time to adjust to the artificial situation of being interviewed or observed; this advances the chance of the development of a trusting relationship. Additionally the participants were promised total anonymity.

A third concern of case studies is whether they provide basis for scientific generalisation. A short answer to that is: case studies are generalisable to theoretical propositions and not to populations or universes (Yin 2009). Walsham supports this statement and adds four types of generalisations: 1) the development of concepts, 2) the generation of theory, 3) the drawing on specific implications and 4) the contribution of rich insight (Walsham 1995). The thesis seeks to provide rich insights, especially about the six practice areas. The propositions are based the detailed descriptions of each case study (see section 6.3). The findings are discussed according to existing literature and on the basis of this generalized according to theory.

The Case Studies and How They Were Conducted

The case studies chosen have some limitations: 1) in terms of boundaries in the understandings provided, 2) accessibility, and 3) in the support they offer each other.

The boundaries of the case studies limited the understandings they provided. The case studies chosen in this thesis both engage in safety-critical software development, which is highly relevant as this condition requires that stringent quality assurance is applied. The case studies answer the research question for this type of cases and cases with similar conditions. Whether the findings will apply completely to cases with other characteristics and other conditions is left unanswered. Case studies of other types of organizations may reveal additional challenges and difficulties than those faced by these organizations; hence, left as implications for future research. However, as these case studies represent the most challenging conditions for meshing agile and documentation-driven methods, the findings represent a minimum of compatibility. The case studies did not provide findings in all nine practice areas, which also serve as a limitation. The thesis is not able to answer how to mesh the practice areas of management strategy, people-issues and culture. How to achieve a mesh on these can therefore in this thesis only be answered based on the theory.

A second limitation was the accessibility offered by the case studies. Gaining access to both case organizations was not a problem; maintaining access and gaining access to all types of data was, however, more difficult. In case study 1, the software organization was very open and the accessibility was high. Broadening the focus to include the customer and the relations between the two, however, limited the access. Neither the software organization, nor the customers wanted to provide information that potentially could harm their trusting relationship. The central customer organization was especially cautious sharing information. This limited the access to the data. In case study 2 gaining access to the software groups was easy as well. Gaining access to people outside the software groups was, however, not possible in neither project A or B. The information was by the management perceived to be too sensitive. In the second phase of case study 2, a new project manager had been assigned. Getting access to him was not possible as he did not approve of the collaboration. Due to the disapproval of the new project manager, the analysis of phase two was not presented and validated on a seminar (like in the first phase). The second validation was therefore only in terms of feedback from the report provided.

In order to support and strengthen the analysis of the data three frameworks have been used; each represents a different perspective on the data. The frameworks provide a theoretical foundation for the analysis; they contribute to connecting the case studies and theory and they strengthen the possibility of generalising the findings. However, frameworks represent a limitation in themselves, as they narrow the focus of the data collection and analysis. In order to compensate for this limitation, the thesis uses both a bottom-up (the research is guided by interesting findings) and a top-down approach (the research is guided by theoretically developed frameworks). The collection of data was furthermore conducted based on semi-structured interview guides allowing the researcher to pursue other interesting emerging topics.

6.5 Future Research

There is still a great need for further research in the area of meshing agile and documentation-driven methods in practice. There is a need to provide software organizations with further information on how compatible the agile and the documentation-driven methods are in practice, how they can be meshed, and strategies for how to overcome the challenges and barriers of meshing the two.

This thesis has focused on how to mesh agile and documentation-driven methods when developing safety-critical software. There is also a need for studying how to do this in a larger variety of organizations. To be able to determine possible differences, it is relevant to include both a high variety in the types of organizations: including large, medium and small organizations, including both public and private organizations and including different types of software development, for example development of enterprise systems, web applications and so on. Such case studies will provide a different type of empirical data. The results will most likely show some differences between these organizations and those organizations developing safety-critical software.

The research approach can also be extended to include a combination of different types of research methods. The framework of Braa and Vidgen (1999) among others includes action cases, that for example could be used to investigate different ways of achieving a mesh or how a meshed software practice could be improved. In such a case study, the framework of the home grounds could provide structure and contribute to an explanation of the software practice and its activities. The framework of the attributes and processes of interorganizational relationships can be used as a foundation for improving the customer relations in a meshed software practice. The knowledge transfer framework can be used to identify the barriers of both the implementation of agile practices, but also the barriers to the diffusion of a meshed software practice within the organization and provide a basis for understanding which actions to undertake to overcome each barrier.

It would be relevant to study each of the six practice areas presented in section 6.3 in more depth.

- A study whose purpose is to understand in greater depth how iterative software development can be conducted within long-term project milestones is highly relevant. The study should also include experiments with different iterations lengths and different considerations as to what can be treated as deliverables at the end of an iteration.
- As the amount of documentation has shown to be the greatest challenge to implementing a meshed practice, research on the minimum amount of documentation required is needed. Furthermore, it is not well understood how to handle these documents in a more agile manner.
- Studying the compatibility on the documentation will also provide knowledge on how to increase the compatibility of the communication and knowledge sharing strategies. It is not understood, what an appropriate mesh between these strategies would be, Hansen et al. (1999) advocate an 80-20% distribution of the strategies, but does this also apply to a meshed practice?
- The challenges identified with meshing the agile and the documentation-driven strategies for handling requirements is also highly connected to the question of how much documentation is required, and how to handle the documentation.
- Further studies on how to mesh the different strategies for customer relations in organizations with direct customer contact is also relevant. Studies on how to handle the issue of not having a real customer associated should also be included, as not all cases have a direct customer and as it has proven difficult to implement a functioning product owner role internally in the organization, thus more knowledge on how to do this is needed.
- Finally, it will be relevant to conduct in-depth studies on how to implement test-driven software development in order to determine their challenges and potential barriers by providing a deeper understanding of this challenge.

7. CONCLUSION

The purpose of this thesis was to answer the research question: how can the agile and the documentation-driven methods be meshed in practice? The research is based on a literature review and two interpretive case studies. The purpose of the literature review was to obtain an overview of the existing literature, providing a theoretical foundation. Both case studies focus on safety-critical software development and by law have to comply with strict quality standards that govern how the software is developed. At the same time both organizations were attempting to achieve a mesh. The first case study showed how an agile software organization attempted to implement a meshed practice by introducing documentation-driven elements. The second case study showed how two software groups in documentation-driven projects attempted to implement a meshed practice by introducing agile elements.

Based on the literature, nine practice areas of meshing were identified (Table 6, section 2.4):

1. management style,
2. customer relations,
3. people-issues,
4. documentation,
5. requirements,
6. development strategy,
7. communication and knowledge sharing,
8. testing and
9. culture

The case studies provided contributions on six of the nine practice areas; the propositions developed on these areas are:

Proposition 1: The practice area of customer relations is difficult to mesh. Implementing documentation-driven customer relations in an agile practice is hard as the level of trust can hinder this. When implementing an agile product owner role in a documentation-driven project highlighting the importance of this role to the management is essential.

Proposition 2: The practice area of documentation is the hardest to mesh. The amount of documentation required by quality standards is not supported by the agile focus on working software. Handling the documents in a light and iterative manner may be difficult in a project controlled by long-term milestones and approval of the documents in the early stages.

Proposition 3: The practice area of requirements is difficult to mesh. The main challenge is the different emphasis on the amount of documentation needed and the difficulties of handling the requirements specifications iteratively.

Proposition 4: The practice area of development strategy can be meshed. One way is to embed an agile, iterative software development strategy in a documentation-driven project. Such a practice has several advantages (such as being forced to break the tasks down) but it does pose some challenges, as the long-term milestones will affect the content of the iterations.

Proposition 5: The practice area of communication and knowledge sharing can be meshed, but combinability has not been determined. Creating a mesh between the agile personalization strategy and the documentation-driven personalization strategy is advocated. A mesh will most likely be dominated by the codification strategy, due to the requirements of the quality standards.

Proposition 6: The practice area of testing is difficult to mesh. Implementing an agile, test-driven practice requires that both developers and the management realize the advantages of test-driven development. In a documentation-driven project education on agile testing is useful.

Together, it demonstrates that the practice areas of the agile and the documentation-driven methods have different levels of compatibility as described in proposition one to six. In total, it is possible to implement a meshed software development practice. Due to the differences of the agile and the documentation-driven methods, the case studies did however not show combinability of the methods in practice. The thesis showed that the agile and documentation-driven methods can be meshed by dividing the software practice into short iterations within long-term milestones. However, the content and the outcome of the iterations are affected by the milestones as the passing of these are dependent on certain activities, for example writing documentation. A software organization can also mesh the agile and the documentation-driven methods by introducing an agile personalised communication and knowledge strategy; this can be implemented by, for example, introducing short, daily meetings. The amount of documentation and the way the requirements are handled are however more difficult to mesh, as the agile methods do not support the amount of documentation required by documentation-driven methods; the documentation of the requirements makes it difficult to handle these in an iterate manner. The study also showed that it can be difficult to implement a customer- and test-driven practice, as these are not prioritised in a documentation-driven practice.

REFERENCES

Aaen, I. (2003). Software process improvement: Blueprints versus recipes. *IEEE Software,* (20:5): 86-93.

Abrahamsson, P., Conboy, K. and Wang, X. (2009). 'Lots done, more to do': the current state of agile systems development research. *European Journal of Information Systems,* (18:4): 281-284.

Agile Alliance, (2011). *The Agile Alliance,* [Homepage of Agile Alliance], [Online]. Available: http://www.agilealliance.org [2011, August, 31].

Anderson, D.J. (2005). Stretching agile to fit CMMI level 3. *In: Proceedings of the Agile Development Conference,* IEEE Computer Society, Denver, p. 193.

Argote, L. and Ingram, P. (2000). Knowledge transfer: A basis for competitive advantage in firms. *Organizational behavior and human decision processes,* (82:1): 150-169.

Avison, D. and Fitzgerald, G. (2006). *Information systems development: methodologies, techniques and tools,* McGraw Hill, Berkshire, Britain.

Avison, D.E. and Fitzgerald, G. (2003). Where now for development methodologies? *Communications of the ACM,* (46:1): 78-82.

Baker, M.J. (2000). Writing a literature review. *The Marketing Review,* (1:2): 219-247.

Baker, S.W. (2005). Formalizing agility: An agile organization's journey toward CMMI accreditation. *In: Proceedings of the Agile Development Conference,* IEEE Computer Society, Denver, CO, USA, p. 185.

Bandara, W., Miskon, S. and Fielt, E. (2011). A systematic, tool-supported method for conducting literature reviews in information systems. *In: Proceedings of the19th European Conference on Information Systems (ECIS 2011),* Helsinki, Finland.

Bang, S., Efsen, S., Hundborg, P., Janum, H., Mathiassen, L. and Schultz, C. (2002). *Kvalitetsstyring i systemudvikling,* Teknisk Forlag, Århus, Denmark.

Batista, J. and Figueiredo, A. (2000). SPI in a very small team: A case with CMM. *Software Process: Improvement and Practice,* (5:4): 243-250.

Beck, K. and Andres, C. (2004). *Extreme programming explained: Embrace change,* Addison-Wesley Professional, USA.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J. & Thomas, D. (2001). *Manifesto for Agile Software Development.* Available: http://agilemanifesto.org [2011, October/9].

Beck, K. and Boehm, B. (2003). Agility through discipline: A debate. *Computer,* (36:6): 44-46.

Benbasat, I., Goldstein, D.K. and Mead, M. (1987). The case research strategy in studies of information systems. *MIS Quarterly,* 369-386.

Beznosov, K. (2003). Extreme Security Engineering: On Employing XP Practices to Achieve 'Good Enough Security' without Defining It. *In: First ACM Workshop on Business Driven Security Engineering (BizSec).* Citeseer, Fairfax, VA, USA.

Beznosov, K. and Kruchten, P. (2004). Towards agile security assurance. *In: Proceedings of the 2004 Workshop on New Security Paradigms,* ACM, Nova Scotia. Canada, p. 47.

Boehm, B. (2002). Get ready for agile methods, with care. *Computer,* (35:1): 64-69.

Boehm, B. and Turner, R. (2005). Management challenges to implementing agile processes in traditional development organizations. *IEEE Software,* (22:5): 30-39.

Boehm, B. and Turner, R. (2004). Balancing agility and discipline: Evaluating and integrating agile and plan-driven methods. *In: Proceedings of the 26th International Conference on Software Engineering,* IEEE Computer Society, Washington, DC, USA, p. 718.

Boehm, B. and Turner, R. (2003a). Observations on balancing discipline and agility. *In: Proceedings of the Agile Development Conference,* IEEE Computer Society, Salt Lake City, Utah, USA, pp. 32-39.

Boehm, B. and Turner, R. (2003b). People factors in software management: Lessons from comparing agile and plan-driven methods. *Crosstalk - The Journal of Defense Software Engineering,* December): 4-8.

Boehm, B. and Turner, R. (2003c). Using risk to balance agile and plan-driven methods. *Computer,* (36:6): 57-66.

Boehm, B.W. and Turner, R. (2003d). *Balancing agility and discipline: A guide for the perplexed,* Addison-Wesley Professional, Boston, USA.

Bos, E. and Vriens, C. (2004). An agile CMM. *In: proceedings of 4th Conference on Extreme Programming and Agile Methods-XP/Agile Universe,* Springer, Calgary, Canada, p. 129.

Boström, G., Wäyrynen, J., Bodén, M., Beznosov, K. and Kruchten, P. (2006). Extending XP practices to support security requirements engineering. *In: Proceedings of the 2006 international workshop on Software engineering for secure systems,* ACM, Shanghai, China, p. 11.

Braa, K. and Vidgen, R. (1999). Interpretation, intervention, and reduction in the organizational laboratory: A framework for in-context information system research. *Accounting, Management and Information Technologies,* (9:1): 25-47.

Brodman, J.G. and Johnson, D.L. (1994). What small business and small organizations say about the CMM: experience report. *In: Proceedings of the 16th international conference on Software engineering,* IEEE Computer Society Press, p. 331.

Brunard, V. and Kleiner, B. (1994). Developing trustful and cooperative relationships. *Leadership and Organization Development Journal,* (15:2): 16-19.

Cawley, O., Wang, X. and Richardson, I. (2010). Lean/agile software development methodologies in regulated environments–state of the art. *Lean Enterprise Software and Systems,* (65:1): 31-36.

Chau, T. and Maurer, F. (2004). Knowledge sharing in agile software teams. *Logic versus approximation,* 173-183.

Chau, T., Maurer, F. and Melnik, G. (2003). Knowledge sharing: Agile methods vs. tayloristic methods. *In: Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on,* IEEE, Los Alamitos, USA, p. 302.

Checkland, P. and Scholes, J. (1990). *Soft Systems Methodology in Action,* Wiley, Chichester.

Chrissis, M.B., Konrad, M. and Shrum, S. (2003). *CMMI guidelines for process integration and product improvement,* Addison-Wesley Longman Publishing Co. Inc., Boston, MA, USA.

Cockburn, A. (2006). *Agile software development: The cooperative game,* Addison-Wesley Professional, Boston, USA.

Cockburn, A. (2005). *Crystal clear,* Addison-Wesley, USA.

Cockburn, A. and Highsmith, J. (2002). Agile software development, the people factor. *Computer,* (34:11): 131-133.

Cohn, M. and Ford, D. (2003). Introducing an agile process to an organization [software development]. *Computer,* (36:6): 74-78.

Conboy, K. (2009). Agility from first principles: reconstructing the concept of agility in information systems development. *Information Systems Research,* (20:3): 329-354.

Crawford, B., Castro, C. and Monfroy, E. (2006). Knowledge management in different software development approaches. *In: Advances in Information Systems, LNCS 4243,* Springer, Izmir, Turkey, p. 304.

Dahlberg, H., Ruiz, F.S. and Olsson, C.M. (2006). The role of extreme programming in a plan-driven organization. *In: The Transfer and Diffusion of Information Technology for Organizational Resilience: IFIP TC8 WG 8.6 International Working Conference,* Springer, Galway, Ireland, p. 291.

Dahlbom, B. and Mathiassen, L. (1993). *Computers in context: The philosophy and practice of systems design,* Wiley-Blackwell, USA.

Das, T.K. and Teng, B.S. (1998). Between trust and control: Developing confidence in partner cooperation in alliances. *Academy of management Review,* (23:3): 491-512.

Das, T. and Teng, B.S. (2001). Trust, control, and risk in strategic alliances: An integrated framework. *Studies,* (22:2): 251-283.

DeMarco, T. (1979). *Structured analysis and system specification,* Yourdon Press, New York, USA.

DeMarco, T. and Boehm, B. (2002). The agile methods fray. *Computer,* (35:6): 90-92.

Dennis, A. and Wixom, B.H. (2000). *Systems analysis and design: An applied approach,* John Wiley & Sons, New York, USA.

Desouza, K.C. (2003). Barriers to effective use of knowledge management systems in software engineering. *Communications of the ACM,* (46:1): 99-101.

Dybå, T. and Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology,* (50:9-10): 833-859.

Elshafey, L.A. and Galal-Edeen, G.H. (2008). Combining CMMI and Agile Methods. *In: Proceedings on the 6th International Conference on Informatics and Systems,* Faculty of Computers and Information, Cairo, Egypt, p. 27.

Esfahani, C., Cabot, J. and Yu, E. (2010). Adopting agile methods: Can goal-oriented social modeling help? *In: Fourth International Conference on Research Challenges in Information Science (RCIS),* IEEE, Nice, France, p. 223.

Fitzgerald, B. (1997). The use of systems development methodologies in practice: a field study. *Information Systems Journal,* (7:3): 201-212.

Fitzgerald, B., Russo, N.L. and Stolterman, E. (2002). *Information systems development: Methods in action,* McGraw-Hill, New York, USA.

Galal-Edeen, G.H., Riad, A.M. and Seyam, M.S. (2007). Agility versus discipline: Is reconciliation possible? *In: International Conference on Computer Engineering & Systems, ICCES'07.* IEEE CNF, Cairo, Egypt, p. 331.

Gilb, T. (2006). Evolutionary project management: Multiple performance, quality and cost metrics for early and continuous stakeholder value delivery. *Enterprise Information Systems VI,* (32:10): 24-29.

Goles, T. and Chin, W.W. (2005). Information systems outsourcing relationship factors: Detailed conceptualization and initial evidence. *ACM SIGMIS Database,* (36:4): 67.

Gupta, K.S. (2008). A comparative analysis of knowledge sharing climate. *Knowledge and Process Management,* (15:3): 186-195.

Haider, S. (2001). *ISO 9001/2000, document development compliance manual, a complete guide and CD-ROM,* St. Lucie Press.

Hansen, M.T., Nohria, N. and Tierney, T. (1999). What's your strategy for managing knowledge? *Harvard business review,* (77:2): 106-116.

Hares, J.S. (1994). *SSADM for the Advanced Practitioner,* John Wiley & Sons, UK.

Heimdahl, M.P.E. (2007). Safety and software intensive systems: Challenges old and new. *In: Future of Software Engineering (FOSE '07),* IEEE Computer Society, Minneapolis, Minnesota, USA, p. 137.

Hilburn, T.B. and Townhidnejad, M. (2000). Software quality: a curriculum postscript? *In: Proceedings of the thirty-first SIGCSE technical symposium on Computer science education* ACM, Austin, Texas, USA, p. 167.

Hirschheim, R.A., Klein, H.K. and Lyytinen, K. (1995). *Information systems development and data modeling: conceptual and philosophical foundations,* Cambridge University Press, Cambridge, UK.

Holmström Olsson, H., Conchúir Ó, E. and Ågerfalk J, P. (2008). Two-stage offshoring: An investigation of the irish bridge. *Management Information Systems Quarterly,* (32:2): 257-279.

Hoyle, D. (2006). *ISO 9000 quality systems handbook,* Butterworth-Heinemann, Oxford, UK.

Huo, M., Verner, J., Zhu, L. and Babar, M. (2004). Software quality and agile methods. *In: Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04),* IEEE Computer Society, Hong Kong, p. 520.

Iivari, J. and Iivari, N. (2010). Organizational culture and the deployment of agile methods: The competing values model view. *In: Proceedings of the 11th International Conference on Agile Software Development (XP2010),* Springer, Trondheim, Norway, p. 203.

Iversen, J., Nielsen, P.A. and Norbjerg, J. (1999). Situated assessment of problems in software development. *ACM SIGMIS Database,* (30:2): 66-81.

Jakobsen, C.R. and Johnson, K.A. (2008). Mature agile with a twist of CMMI. *In: Agile, 2008. AGILE'08. Conference,* IEEE Computer Society, Toronto, Canada, p. 212.

Kähkönen, T. and Abrahamsson, P. (2004). Achieving CMMI level 2 with enhanced extreme programming approach. *In: Proceedings of the 5th International Conference, PROFES 2004,* Springer, Kansai Science City, Japan, p. 378.

Kautz, K. and Thaysen, K. (2001). Knowledge, learning and IT support in a small software company. *Journal of Knowledge Management,* (5:4): 349-357.

Kelly, D.P. and Culleton, B. (1999). Process improvement for small organizations. *Computer,* (32:10): 41-47.

Kern, T. (1997). The gestalt of an information technology outsourcing relationship: An exploratory analysis. *In: Proceedings of the Eighteenth International Conference on Information Systems,* Association for Information Systems, Atlanta, USA, p. 37.

Kniberg, H. (2007). *Scrum and XP from the trenches: how we do scrum,* Lulu Com.

Kruchten, P. (2004). *The rational unified process: an introduction,* Addison-Wesley Professional, Boston, USA.

Larman, C. (2004). *Agile and iterative development: a manager's guide,* Addison-Wesley Professional, Boston, USA.

Laurila, P. (2004). Are CMM and agile methods really compatible? *In: T-76.650 Seminar in software engineering,* Software Business and Engineering Institute, Helsinki, Finland, p. 1.

Lee, I., Pappas, G.J., Cleaveland, R., Hatcliff, J., Krogh, B.H., Lee, P., Rubin, H. and Sha, L. (2006a). High-confidence medical device software and systems. *Computer,* (39:4): 33-38.

Lee, O.K.D., Banerjee, P., Lim, K.H., Kumar, K., Hillegersberg, J. and Wei, K.K. (2006b). Aligning IT components to achieve agility in globally distributed system development. *Communications of the ACM,* (49:10): 48-54.

Lindvall, M. and Rus, I. (2002). Knowledge management in software engineering. *IEEE Software,* (19:3): 26-38.

Lyytinen, K. and Robey, D. (1999). Learning failure in information systems development. *Information Systems Journal,* (9:2): 85-101.

Madsen, S., Kautz, K. and Vidgen, R. (2006). A framework for understanding how a unique and local IS development method emerges in practice. *European Journal of Information Systems,* (15:2): 225-238.

Mahanti, A. (2004). Challenges in enterprise adoption of agile methods - A survey. *Journal of Computing and Information Technology,* (14:3): 197-206.

Marçal, A.S.C., de Freitas, B.C.C., Soares, F.S.F., Furtado, M.E.S., Maciel, T.M. and Belchior, A.D. (2008). Blending Scrum practices and CMMI project management process areas. *Innovations in Systems and Software Engineering,* (4:1): 17-29.

Mathiassen, L. (1982). *Systemudvikling og systemudviklingsmetode,* Datalogisk Afdeling, Matematisk Institut, Aarhus Universitet, Århus, Denmark.

Mathiassen, L. and Pries-Heje, J. (2006). Business agility and diffusion of information technology. *European Journal of Information Systems,* (15:2): 116-119.

Mathiassen, L. and Purao, S. (2002). Educating reflective systems developers. *Information Systems Journal,* (12:2): 81-102.

McAvoy, J. and Butler, T. (2009). A Failure to Learn in a Software Development Team: The Unsuccessful Introduction of an Agile Method. *In: Proceedings of the 16th International Conference on Information Systems Development (ISD2007),* Springer, Cairn, Ireland, p. 1.

McDermott, R. and O'Dell, C. (2001). Overcoming cultural barriers to sharing knowledge. *Journal of Knowledge Management,* (5:1): 76-85.

McMichael, B. and Lombardi, M. (2007). ISO 9001 and Agile development. *In: Proceedings of the AGILE 2007,* IEEE Computer Society, Washington DC, USA, pp. 262.

Misra, S.C., Kumar, V. and Kumar, U. (2010). Identifying some critical changes required in adopting agile practices in traditional software development projects. *International Journal of Quality & Reliability Management,* (27:4): 451-474.

Moe, N.B., Dingsøyr, T. and Dybå, T. (2010). A teamwork model for understanding an agile team: A case study of a Scrum project. *Information and Software Technology,* (52:5): 480-491.

Muhr, T. (1991). ATLAS/ti—A prototype for the support of text interpretation. *Qualitative Sociology,* (14:4): 349-371.

Murru, O., Deias, R. and Mugheddue, G. (2003). Assessing XP at a European internet company. *Software, IEEE,* (20:3): 37-43.

Myers, M.D. and Newman, M. (2007). The qualitative interview in IS research: Examining the craft. *Information and Organization,* (17:1): 2-26.

Namioka, A. and Bran, C. (2004). eXtreme ISO?!? *In: Conference on Object Oriented Programming Systems Languages and Applications,* ACM, New York, USA, p. 260.

Navarrete, F.J., Botella, P. and Franch, X. (2006). An Approach to Reconcile the Agile and CMMI Contexts in Product Line Development. *In: International Workshop on Agile Product Line Engineering.*

Nawrocki, J., Jasiñski, M., Walter, B. and Wojciechowski, A. (2002). Extreme programming modified: embrace requirements engineering practices. *In: 10th IEEE Joint International Requirements Engineering Conference, RE,* IEEE Computer Society, Essen, Germany, p. 303.

Nawrocki, J., Olek, L., Jasinski, M., Paliswiat, B., Walter, B., Pietrzak, B. and Godek, P. (2006). Balancing agility and discipline with xprince. *In: Rapid Integration of Software Engineering Techniques, Second International Workshop, RISE 2005,* Springer, Heraklion, Crete, Greece, pp. 266.

Nawrocki, J.R., Walter, B. and Wojciechowski, A. (2002). Comparison of CMM level 2 and eXtreme programming. *In: Software Quality ECSQ 2002, the 7th European Conference on Software Quality,* Springer, Hensinki, Finland, p. 288.

Nerur, S. and Balijepally, V.G. (2007). Theoretical reflections on agile development methodologies. *Communications of the ACM,* (50:3): 79-83.

Nerur, S., Mahapatra, R.K. and Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM,* (48:5): 73-78.

Ngwenyama, O. and Nielsen, P.A. (2003). Competing values in software process improvement: an assumption analysis of CMM from an organizational culture perspective. *IEEE Transactions on Engineering Management,* (50:1): 100-112.

Nielsen, P.A. (1991). *Learning and using methodologies in information systems analysis and design,* Institute for Electronic Systems, Department of Mathematics and Computer Science, Aalborg, Denmark.

Nonaka, I. (1994). A dynamic theory of organizational knowledge creation. *Organization Science,* (5:1): 14-37.

O'Dell, C.S., O'Dell, C., Grayson, C.J. and Essaides, N. (1998). If only we knew what we know: The transfer of internal knowledge and best practice. *California Management Review,* (40:3): 154-174.

Omland, H.O. (2009). The Relationships Between Competence, Methods, and Practice in Information Systems Development. *Scandinavian Journal of Information Systems,* (21:2): Article 5.

Opelt, K. and Beeson, T. (2008). Agile teams require agile QA: How to make it work, an experience report. *In: Agile, 2008. AGILE'08.* IEEE Computer Society, Toronto, p. 229.

Oxford Advanced Learner's Dictionary. (2011). *Oxford Advanced Learner's Dictionary,* [Homepage of Oxford University Press], [Online]. Available: http://www.oxfordadvancedlearnersdictionary.com/ [2011, September, 22].

Paetsch, F., Eberlein, A. and Maurer, F. (2003). Requirements engineering and agile software development. *In: Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises,* Citeseer, Linz, Austria, p. 308.

Paulk, M.C. (2002). Agile methodologies and process discipline. *Crosstalk - The Journal of Defense Software Engineering,* (1:1): 15-18.

Paulk, M.C. (2001). Extreme programming from a CMM perspective. *IEEE Software,* (18:6): 19-26.

Pettigrew, A.M. (1990). Longitudinal field research on change: theory and practice. *Organization Science,* (1:3): 267-292.

Pikkarainen, M., Salo, O. and Still, J. (2005). Deploying agile practices in organizations: A case study. *In: EuroSPI 2005, LNCS 3792,* Springer, Berlin, p. 16.

Poppendieck, M. (2007). Lean software development. *In: Companion to the proceedings of the 29th International Conference on Software Engineering,* IEEE Computer Society, Minneapolis, Minnesota, p. 165.

Poppendieck, M. and Poppendieck, T. (2003). *Lean software development: An agile toolkit,* Addison-Wesley Professional, USA.

Pressman, R.S. (2000). *Software engineering - A practitioner's approach,* McGraw-Hill Publishing Company, UK.

Qumer, A. and Henderson-Sellers, B. (2008). An evaluation of the degree of agility in six agile methods and its applicability for method engineering. *Information and Software Technology,* (50:4): 280-295.

Reifer, D.J. (2003). XP and the CMM. *IEEE Software,* (20:3): 14-15.

Riege, A. (2005). Three-dozen knowledge-sharing barriers managers must consider. *Journal of Knowledge Management,* (9:3): 18-35.

Rising, L. and Janoff, N.S. (2002). The Scrum software development process for small teams. *Software, IEEE,* (17:4): 26-32.

Rönkkö, M., Jarvi, A. and Makela, M.M. (2008). Measuring and Comparing the Adoption of Software Process Practices in the Software Product Industry. *In: Making Globally Distributed Software Development a Success Story, The International Conference on Software Process, ICSP 2008,* Springer, Leipzig, Germany, p. 407.

Sabherwal, R. (1999). The role of trust in outsourced IS development projects. *Communications of the ACM,* (42:2): 86.

Sambamurthy, V., Bharadwaj, A. and Grover, V. (2003). Shaping agility through digital options: Reconceptualizing the role of information technology in contemporary firms. *MIS Quarterly,* (27:2): 237-263.

Sarker, S., Munson, C.L., Sarker, S. and Chakraborty, S. (2009). Assessing the relative contribution of the facets of agility to distributed systems development success: An analytic hierarchy process approach. *European Journal of Information Systems,* (18:4): 285-299.

Schwaber, K. and Beedle, M. (2001). *Agile software development with Scrum,* Prentice Hall, Upper Saddle River, New Jersey, USA.

Senapathi, M. (2010). Adoption of software engineering process innovations: The case of agile software development methodologies. *In: Agile Processes in Software Engineering and Extreme Programming, 11th International Conference, XP 2010,* Springer, Trondheim, Norway, p. 226.

Sidky, A., Arthur, J. and Bohner, S. (2007). A disciplined approach to adopting agile practices: the agile adoption framework. *Innovations in Systems and Software Engineering,* (3:3): 203-216.

Stålhane, T. and Hanssen, G.K. (2008). The application of ISO 9001 to agile software development. *In: Product-Focused Software Process Improvement, the 9th International Conference, PROFES 2008,* Springer, Monte Porzio Catone, p. 371.

Stapleton, J. (1999). DSDM: Dynamic Systems Development Method. *In: Technology of Object-Oriented Languages and Systems,* Published by the IEEE Computer Society, Nancy, France, p. 406.

Stephenson, Z.R., McDermid, J.A. and Ward, A.G. (2006). Health modelling for agility in safety-critical systems development. *In: System Safety, 2006, The First International Conference on Institution of Engineering and Technology,* Citeseer, London, UK, p. 260.

Stober, T. and Hansmann, U. (2010). *Traditional software development,* Springer, Heidelberg, Germany.

Strauss, A.L. and Corbin, J. (1990). *Basics of qualitative research: Grounded theory procedures and techniques,* Sage Newbury Park, USA.

Szulanski, G. (2000). The process of knowledge transfer: A diachronic analysis of stickiness. *Organizational behavior and human decision processes,* (82:1): 9-27.

Taylor, P., Greer, D., Sage, P., Coleman, G., McDaid, K., Lawthers, I. and Corr, R. (2006). Applying an agility/discipline assessment for a small software organisation. *In: Product-Focused Software Process Improvement,* Springer, Amsterdam, Netherlands, p. 290.

Tellez-Morales, G. (2009). XP practices: A successful tool for increasing and transferring practical knowledge in short-life software development projects. *In: Agile Processes in Software Engineering and Extreme Programming,* Springer, Pula, Sardinia, Italy, p. 155.

The International Society for Pharmaceutical Engineering. (2010). *GAMP Publications,* [Homepage of Engineering Pharmaceutical Innovation], [Online]. http://www.ispe.org/index.php/ci_id/11614/la_id/1.htm [2011, January, 14].

Theunissen, W.H., Kourie, D.G. and Watson, B.W. (2003). Standards and agile software development. *In: Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology,* South African Institute for Computer Scientists and Information Technologists, Republic of South Africa, Johannesburg, p. 178.

Tjørnehøj, G. (2006). Improving agile software Practice. *In: Proceedings of the 29th Information Systems Research Seminar in Scandinavia,* Citeseer, Helsingør, Danmark, pp. 1-6.

Turner, R. (2002). Agile development: Good process or bad attitude? *In: Product Focused Software Process Improvement, 4th International Conference PROFES 2002,* Springer, Rovaniemi, Finland, p. 134.

Turner, R. and Jain, A. (2002). Agile meets CMMI: Culture clash or common cause? *In: Extreme Programming and Agile Methods, Second XP Universe and First Agile Universe Conference,* Springer, Chicago, IL, USA, p. 153.

U.S. Department of Health and Human Services. (2010). *FDA U.S. Food and Drug Administration,* [Homepage of U.S. Department of Health and Human Services], [Online]. Available: http://www.fda.gov/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/ucm073778.htm [2011, January, 14].

Vinekar, V., Slinkman, C.W. and Nerur, S. (2006). Can agile and traditional systems development approaches coexist? An ambidextrous view. *Information Systems Management,* (23:3): 31-42.

Vogel, D.A. (2006). Agile Methods: Most are not ready for prime time in medical device software design and development. *DesignFax Online,* 1-6.

Walsham, G. (2006). Doing interpretive research. *European Journal of Information Systems,* (15:3): 320-330.

Walsham, G. (1995). Interpretive case studies in IS research: nature and method. *European journal of information systems,* (4:2): 74-81.

Webster, J. and Watson, R.T. (2002). Analyzing the past to prepare for the future: Writing a literature review. *MIS Quarterly,* (26:2): 13-23.

Wright, G. (2003). Achieving ISO 9001 certification for an XP company. *In: Extreme Programming and Agile Methods, Third XP Agile Universe Conference,* Springer, New Orleans, USA, p. 43.

Yin, R.K. (2009). *Case study research: Design and methods,* Sage Publications Inc, USA.

Yourdon, E. (1989). *Modern structured analysis,* Prentice Hall PTR, USA.

Zaheer, A., McEvily, B. and Perrone, V. (1998). Does trust matter? Exploring the effects of interorganizational and interpersonal trust on performance. *Organization Science,* (9:2): 141-159.

Zanatta, A.L. and Vilain, P. (2006). Extending an agile method to support requirements management and development in conformance to CMMI. *HIFEN,* (30:58): 25-31.

APPENDIX: THE PAPERS

Appendix A (Paper 1):

Heeager, L.T (Presented 2011) "The Agile and the Disciplined Software Approaches: Combinable or Just Compatible?", Information Systems Development (ISD2011), Edinburgh, Scotland.

Appendix B (Paper 2):

Heeager, L.T and Tjørnehøj, G. (2011) "Adopting Quality Assurance Technology in Customer-vendor Relationships: A Case Study of how Organizational Relationships Influence the Process", Information Systems Development (ISD2009), pp. 235-246, Nanchang, China.

Appendix C (Paper 3):

Heeager, L.T and Nielsen P.A. (2009) "Agile Software Development and its Compatibility with a Document-Driven Approach? A Case Study", Australasian Conference on Information Systems (ACIS2009), pp. 205-214, Melbourne, Australia.

Appendix D (Paper 4):

Heeager, L.T (Resubmitted 2011, accepted with minor revisions) "Introducing Agile Practices in a Documentation-Driven Software Development Process: A Case Study", Journal of Information Technology Case and Application Research.

Appendix E (Paper 5):

Heeager, L.T and Nielsen P.A. (Submitted 2011, status: revise and resubmit) "Barriers to Adoption of Agile Development Practices: A Knowledge Transfer Framework", Information Systems Journal.

Presented at the 20th International Conference on Information Systems Development

The Agile and the Disciplined Software Approaches: Combinable or just Compatible?

Lise Tordrup Heeager

Aalborg University, Aalborg Ø, Denmark

e-mail: liseh@cs.aau.dk

Abstract Offhand the agile and the disciplined software development approaches seem contradicting. More and more software development organizations however strive at implementing an agile software development approach while still being compliant to a quality assurance standard. Researchers are discussing the combinability and compatibility of these two approaches. Through a review of the literature the purpose of this paper is to determine whether the agile and the disciplined software development approaches are combinable or just compatible, in particular to identify the main challenges of using an agile software development approach in a disciplined setting. The review shows that the agile and the disciplined approaches are compatible, but not combinable. It is possible to implement agile practices and principles in a development process compliant with a quality standard, but the regulations of the standard makes it impossible to implement a full agile software development process without compromising the agility. The main challenges, when balancing the agility and discipline in a project, are: how to determine the right level of documentation and how to overcome the differences in the way requirements are handled.

Keywords agile software development approaches, disciplined software development approaches, compatibility, combinability, XP, Scrum, CMMI, ISO

1 Introduction

In information systems development research agile software development is a popular topic. This paper uses the terminologies of agile and disciplined software development approaches adopted from Boehm [1, 2]. The agile and the traditional, disciplined software methodologies are two approaches to describe processes for developing software [3]. The agile software development methodologies (e.g. XP and Scrum) promise a way to deliver software without excessive cost. On the contrary disciplined software development processes [2] (e.g. CMMI and the ISO standards) are well-defined and proven, but require a lot of effort [4].

Offhand the two seem contradicting [5, 6], but several researchers agree that a software project needs both agility and discipline [1, 7, 8]. Neither the agile nor the disciplined approaches provide the ultimate approach, both has short-comings and pitfalls [1]. The approach of the disciplined methodologies (plan everything and follow the plan) works well for stable and less complex projects, but they lack the ability of handling change and complexity. The agile methodologies embrace change [9], but offhand they lack the ability of quality assurance.

Software development organizations are increasingly interested in the possibility of adopting agile approaches [10]; even organizations that have been employing process standards are now increasingly interested in the possibility of adopting agile approaches [11]. Rapid change and increasing software criticality drive organizations to balance the agility and discipline [2].

There are on-going debates whether the quality of the products of the agile approaches are satisfactory [12] and some projects e.g. safety-critical projects require standards to be followed when developing software [13, 14, 15]. Since the agile and the disciplined approaches are grounded in opposing concepts, balancing the two is not straightforward [16, 17, 18]. Some organizations attempt to use an agile software development approach and at the same time comply with a quality assurance standard [18, 19], however it is not well understood how to do this the in practice [20. 21, 22]. This has let research to focus on the compatibility and combinability of the agile and the disciplined approaches [23, 24].

With the research question: “are the agile and disciplined approaches combinable or just compatible?” in mind, the purpose of the paper is to give an overview of the combinability and/or the compatibility of the agile and disciplined approaches, and give a more detailed picture of the challenges facing an organization trying to use an agile software development approach in a disciplined setting. This is done through a review of 79 papers dealing with the subject of the combinability or compatibility of the agile and disciplined approaches. This review is relevant to researchers who wish to stay up to date with the state of research on the combinability and/or compatibility of agile and disciplined approaches and to practitioners who wish to implement an agile software development process in practice and at the same time comply with a quality assurance standard.

The paper is structured as following: The research approach is outlined in section 2. Then the analysis on compatibility and combinability is presented in section 3. In section 4 the results of the analysis is discussed also including a discussion of the guidelines and the challenges identified in the literature when trying to obtain compatibility between the agile and disciplined approaches. The analysis results in four propositions also presented in section 4. Finally the results are concluded on in section 5.

2 Research Approach

Previous research on the combinability and/or compatibility of the agile and the disciplined approaches was reviewed. The papers were found through a comprehensive search conducted in three steps (summarized in figure 2.1) [25]. First, the web was searched using relevant keywords. The keywords contained different combinations of names of process standards and agile methodologies and more general keywords as ‘compatibility and agile’ or ‘combine and agile’. In parallel ranked journals and relevant conference proceedings were searched. In the second step a backward search (following references of identified papers) and a forward search (used Google Scholar to find relevant papers citing the identified papers) were done iteratively. At this point the relevance of the papers was determined by reading the titles and abstract, so far the search had resulted in 88 papers. Each of these was read, nine of them were discarded as they lacked relevance, either because the combinability and/or compatibility of the agile and the disciplined approaches was a very small and insignificant part of the paper or the fact that the paper only was dealing with either agile or disciplined approaches and not both, leaving 79 papers for further analysis (see reference list in appendix A).

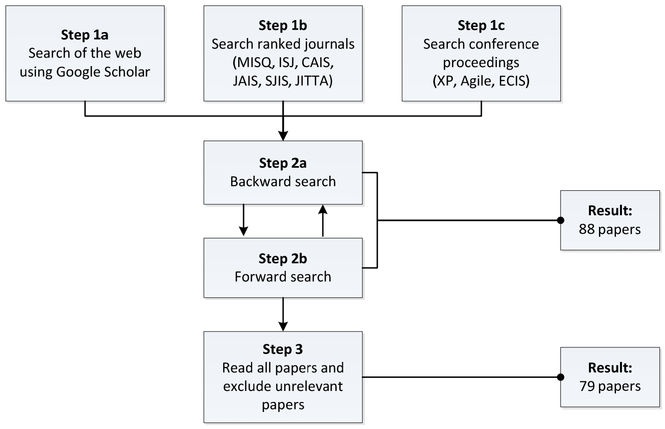

Step 1a
Search of the web using Google Scholar
Step 1b
Search ranked journals (MISQ, ISJ, CAIS, JAIS, SJIS, JITTA)
Step 1c
Search conference proceedings (XP, Agile, ECIS)
Step 2a
Backward search
Step 2b
Forward search
Result:
88 papers
Step 3
Read all papers and exclude unrelevant papers
Result:
79 papers

Figure 2.1. The search strategy.

Conventional content analysis [26] was used as the primary analysis strategy (the analysis strategy is summarized in figure 2.2). This strategy was chosen as existing theory and research literature on the combinability and/or compatibility of the agile and the disciplined approaches is limited. To structure the analysis each of the papers was coded using Atlas.ti V6 [27]. The process consisted of three linear steps – each with a separate result. First, the papers were coded using open coding [28]. The focus when reading and coding the papers was on the combinability/compatibility and non combinability/incompatibility of the agile and the disciplined approaches, on guidelines to overcome the challenges when using an agile software development approach in a disciplined setting and on the strengths and weaknesses of such a practice. This was done in order to get an overview of the content of the previous research. The first step resulted in 59 different codes applied to 1038 quotations. The second step was to categorize the codes, thus creating coding schemes. The resulting coding schemes can be seen in appendix B. One table for each type of codes: general codes, codes describing combinability/compatibility, codes describing non combinability/incompatibility, codes describing guidelines of how to use an agile software development approach in a disciplined setting and codes describing strengths and weaknesses of the agile, the disciplined and the approaches attempting to balance agility and discipline. In the third step the codes and quotes from step 1 were revisited and recoded according to the coding schemes containing 19 categories with 71 codes. This resulted in 1084 quotations.

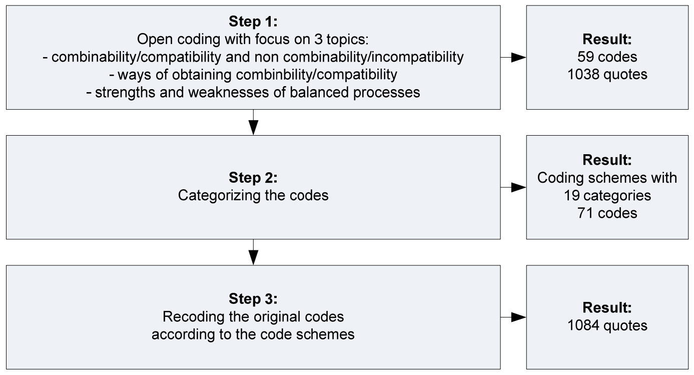

Step 1:
Open coding with focus on 3 topics:
- combinability/compatibility and non combinability/incompatibility
- ways of obtaining combinbility/compatibility
- strengths and weaknesses of balanced processes
Result:
59 codes
1038 quotes
Step 2:
Categorizing the codes
Result:
Coding schemes with
19 categories
71 codes
Step 3:
Recoding the original codes according to the code schemes
Result:
1084 quotes

Figure 2.2. The analysis strategy.

3 Analysis: combinability and compatibility

To be combinable or compatible are two different things. In order to determine the combinability and compatibility of the agile and the disciplined approaches these two terms have to be defined. This is done with a starting point in the dictionary interpretation of these key terms.

To combine is “to join two or more things or groups together to form a single one” [29], meaning that you are able to do two things at the same time without compromising either. To be compatible is to be “able to exist together” [29], meaning that something is able to work at the same time as something else. Thus, combinability is a boolean concept while compatibility can be graded.

The interpretations of the two key terms used in this paper in the context of determining the compatibility and combinability of the agile and the disciplined approaches are: 1) The agile and the disciplined software approaches are combinable if it is possible to use an agile software development approach and at the same time comply with a quality assurance standard without compromising the agility of the agile approach. 2) The agile and the disciplined software development approaches are compatible if it is possible to use some of the agile practices and principles and at the same time be able to comply with a quality assurance standard

The literature review shows that previous research does not distinguish between combinability and compatibility of the agile and the disciplined approaches. Several papers state that the agile and disciplined approaches are combinable, however when reading into the analysis and results presented in the papers adopting the definitions of this paper the conclusion should instead be that the agile and the disciplined approaches are compatible, but not combinable (e.g. S10, S12, S26, S75).

3.1 The compatibility of the agile and the disciplined approaches

Offhand, the agile approaches seem in conflict with the disciplined approaches, but several researchers agree that it is possible to use (some of) the practices and principles of agile software development process and at the same time comply with a quality standard. The majority of the papers reviewed (e.g. S3, S10, S23, S24, S25, S36, S46, S61, S65, S69, S70, S79) state that the agile and the disciplined approaches (to some degree) are compatible. Most papers engage themselves in determining the overall compatibility or incompatibility of the agile and the disciplined approaches, some of these engage in comparing specific methodologies, mainly using XP or Scrum from the agile methodologies and CMMI and ISO from the disciplined methodologies.

Thirty-nine of the papers reviewed outline which agile and disciplined approaches they compare. These are shown in table 3.1. The relationship between XP and CMMI is the one most analyzed. Scrum comes in second among the agile approaches, whereas ISO is the process model second most used. Five papers not only analyze the relationship of an agile and a disciplined approach but also mixed practices from two or more agile approaches. Vriens (S74) not only used a mix of XP and SCRUM but also a mix of CMMI and ISO.

In a point-counter point paper, Beck and Boehm agree that agility and discipline are not opposites (S4). In another paper Boehm also states that even though advocates of the agile and the disciplined approaches consider them opposites, including parts of both in a project can be advantageous (S7) and Glass (S21) points out that “one-size-fits-all” approaches do not work. Mahnic and Zabkar (S34, S35) conclude that it is possible to build software process that balances agility and discipline. Glazer (S22) concludes that the agile approaches and CMMI complete each other and can give fast, affordable, visible and long-term benefits. Through a comparison of the waterfall model and agile approaches, Huo et al. (S25) find that agile approaches contain QA practices, and that these occur more frequently than in the waterfall model. Baker (S3) states that the agile approaches are focused and comprehensive and according to Beck (S4), XP is disciplined, as it provides a clear picture of what activities to apply.

DeMarco and Boehm (S15) support this statement by adding that XP involves more planning than CMM organizations.

	XP	Scrum	RUP	XP/Scrum	Sum
ISO	(S42, S43, S48, S67, S77)	-	-	(S41)	6
CMMI	(S2, S3, S18, S28, S30, S40, S45, S47, S49, S51, S54, S55, S60, S62, S75)	(S16, S26, S27, S34, S35, S38, S39, S66, S78)	(S33, S37)	(S1, S12, S61)	29
CMMI/ISO	(S14)	-	-	(S74)	2
FDA	(S76)	(S24)	-	-	2
Sum	22	10	2	5	

Table 3.1. Comparison of the agile and the disciplined approaches in the reviewed papers.

According to the reviewed papers, using an agile approach in a disciplined setting makes it possible to take advantage of the strengths and compensate for the weaknesses of the two (S10, S19, S44). Turner and Jain (S70) state that the agile and the disciplined approaches have much in common and their strengths and weaknesses are complementary, while Boehm and Turner (S9, S11) present a risk-based method for developing balanced strategies.

However, only few papers deal with the strengths and weaknesses of processes balancing agility and discipline and those who do, only point out the strengths. The disciplined approaches support the agile approaches by providing a disciplined framework (S17), while agile methods ensure that processes are implemented efficiently while embracing change (S14, S26, S66), this balances adaptability and predictability in order to better serve customer needs (S27). The disciplined approaches are also helpful for the agile approaches, as they are able to help identify the shortcomings of the agile approaches (S18), support the decision of which processes to address (S26, 66), provide structure to help ensure your agile processes are followed (S41) and reduce the risk in agile development (S17).

The researchers disagree on how compatible the agile and the disciplined approaches are. The majority of the papers reviewed conclude that it is possible for a company using an extended agile approach to comply with process standards such as ISO and CMMI level 2 or 3. In the following subsections the compatibility and/or combinability of the agile approaches and respectively the ISO standards and the CMM/CMMI will be analysed.

3.2 Agile software methodologies and ISO

The ISO standards: ISO 9000 and ISO 9001 are those mostly used for analysis by researchers dealing with agile software development methodologies and ISO (S41, S42, S43, S48, S64, S77). Three papers do not specify which agile methodology they use for analysis while the remaining five papers either use Scrum or XP (see table 3.2).

	XP	Scrum	Agile	Sum
ISO 9000	-	(S43, S48)	-	2
ISO 9001	(S42)	(S77)	(S41, S64)	4
ISO/IEC15504 (SPICE)	-	-	(S29)	1
ISO/IEC12207/1995	(S67)	-	-	1
Sum	2	3	3	

Table 3.2. Comparison of agile methods and ISO in the reviewed papers.

Agile approaches can be adapted to ensure compatibility with ISO standards (S67). Based on empirical evidence Wright (S77) concludes that companies using XP can meet the requirements of

ISO9001. McMichael and Lombardi (S41) conclude that ISO9001:2000 is able to help ensure the agile processes are followed. Melis (S42) concluded that several requirements of ISO9001:2000 are implemented by the existing tools in XP project management, while Stålhane and Hanssen (S64) suggest few changes to both the ISO standard and the agile approaches. Lami and Falcini (S29) showed that ISO/IEC15504 in principle is applicable to agile contexts, but that in practice, problems may occur e.g. in creating an agile process reference model and in order to find an assessor that has experience from software development in agile contexts.

3.3 Agile software methodologies and CMM/CMMI

Several researchers of the papers (18 of the review papers) compare the components of the CMM (S30, S37, S40, S55) or CMMI (S1, S12, S16, S17, S18, S30, S38, S39, S40, S45, S51, S66, S70, S78) process model with agile methods, such as XP and Scrum. Table 3.3 and table 3.4 give an overview of which agile methods are compared with the generic practices and process areas of respectively CMMI and CMM.

	XP	Scrum	RUP	Agile	Sum
CMMI generic practices	(S18)	(S16, S18, S66)	-	(S1, S70)	6
CMMI process areas	(S12, S18, S30; S40, S45, S51)	(S12, S18, S38, S39, S78)	-	(S1, S17, S44, S70)	15
Sum	7	8	0	6	

Table 3.3. Comparison of agile methods and CMMI in the reviewed papers.

	XP	Scrum	RUP	Agile	Sum
CMM generic practices	-	-	-	-	0
CMM process areas	(S30, S40, S55)	-	(S37)	-	4
Sum	3	0	1	0	

Table 3.4. Comparison of agile methods and CMM in the reviewed papers.

An analysis of the results on the comparison of agile methods with the process areas of CMMI shows that the researchers agree that CMMI level 2 is largely compatible with the agile approaches, CMMI level 3 are partially compatible with agile approaches while CMMI level 4 and 5 are less compatible with agile approaches. There does not seem to be a big difference in the compatibility of CMMI and respectively XP and Scrum.

Fifteen papers state that the agile approaches are compatible with CMMI level 2 or 3. Marcal et al. (S38, S39) say that it is possible to reach CMMI level 2 without compromising the agility. Alegria and Bastarrica (S1) state that each process area needs to be defined explicitly by the organization and complemented with elements obtained from other sources. Baker (S3), Bos and Vriens (S12) and Kähkönen and Abrahamsson (S28) conducted case studies of companies complying with CMMI level 2 using agile approaches. Santana et al. (S62) conducted studies of two companies complying with respectively CMMI level 2 and 3, respectively using XP and Scrum while Manzoni et al. (S37) concludes that reaching CMMI level 3 using RUP is possible. Rönkkö et al. (S61) found that the process areas of CMMI level 2 and 3 are adopted in parallel and can be supported by the use of agile approaches. Jakobsen and Johnson (S26) recommend extending the agile approaches using the mandatory goals and expected practices of CMMI level 2 and 3. Laurila (S30) concluded that CMM and agile methods are practice compatible, but not idea compatible.

Several researchers state that agile methods do not address the CMMI level 4 or 5 in their original state (e.g. S8, S30, S40, S55), but only one paper states, that it is not possible to reach the highest maturity levels when being agile (S18). Few results show that it is possible to comply with CMMI level 4 or 5. Through an analysis of XP and CMM, Martinsson (S40) found it possible and advantageous to use XP to reach the highest CMM level. At Microsoft they created an agile life cycle complying with CMMI level 5 (S2) and at Systematic they used Scrum to reach the highest level (S66, S77).

The disciplined approaches tell what to do, while the agile approaches tell how to do it (S16). CMMI is a maturity model while agile is a development philosophy (S31).

3.4 Agile approaches for developing safety-critical software

Another ongoing discussion in the reviewed papers is whether or not agile approaches are useful when developing safety-critical software (S32). Safety-critical software is software in which failure can result in direct injury to humans or cause severe economic damage [30]. In security engineering the home grounds of the agile and disciplined approaches are considered far from each other as a high level of discipline is needed throughout the development. The process standards for safety-critical software are often more strict than e.g. ISO9001 (S77) as they require a robust development process that ensures the quality and safety of the product (S76).

Even though developers of safety-critical software are experimenting with agile approaches (S65, S73) only two case studies of the reviewed papers included a company developing a safety-critical product (S24, S76). They use elements of respectively Scrum and XP while complying with the US Food and Drug Administration's (FDA) standard for software development. These cases are proof that it is possible to implement agility in a safety-critical software development process. This proof is supported by other researchers. According to Beznosov and Kructhen (S6), pair programming naturally facilitates internal design and code review, and motivates developers to follow coding standards. Wäyrynen et al. (S75) also conclude that XP is aligned with security engineering.

However the review also shows consensus among the researchers on the fact that agile approaches in their pure form are not suited for developing safety-critical software. Beznosov (S5) introduces eXtreme Security Engineering (XSE), an application of XP practices to security engineering. Boström et al. (S13) also proposes extending XP practices, while Siponen et al. (S63) give an example of how to add security techniques to agile approaches in general.

The researchers have different advice on how to fit the agile methods. For example Pohjola (S58) concludes that the test-driven development practice of XP is the key. Lindvall et al. (S32) gathered the experiences on agility at a workshop. They found that safety-critical projects can be conducted using agile approaches, the key is to make the performance requirements explicit and plan proper levels of testing early in the process. In order to introduce agility in the development processes of safety-critical software in general, a body of evidence that agile approaches provide secure software is needed (S73).

4 Discussion

Addressing the research question this section discusses the combinability and compatibility of the agile and the disciplined approaches. It discusses the guidelines and challenges identified in the literature on obtaining compatibility between the two approaches. Furthermore four propositions created based on the analysis and discussion is presented.

4.1 Compatibility, not combinability

The analysis showed that the agile and the disciplined approaches are highly compatible, that is, in practice it is possible to implement several agile practices and principles in a software development process and at the same time be able to comply with a quality assurance standard. The agile and the disciplined approaches are however too different to be combined, meaning, that the agility of a project and/or the practices and principles of an agile software development process will be harmed in some degree by the regulations of a disciplined quality standard or a process model.

Proposition 1: The agile and the disciplined approaches are compatible, but not combinable.

Taking into consideration that the agile software approaches were developed as a counter-point to the heavy-weighted traditional, disciplined software approaches, this proposition seem natural but however important for both practitioners and researchers. It is important that practitioners understand the consequences of introducing a quality standard in their agile process or that it may be possible to heighten the agility of a software development process complying with a quality standard. For further research it is important to understand the difference between combinability and compatibility when comparing an agile and a disciplined approach.

4.2 Obtaining compatibility

Several researchers focus on how to extend either the agile approaches, the disciplined approaches or both of them in order to make them compatible. Most researchers focus on how to extend the agile approaches where as only two see a need in extending the disciplined approaches. Researchers have different ways of extending the approaches.

Four papers make suggestions on extending XP. Beznosov (S5) focuses on extending XP for security engineering. Nawrocki (S45) proposed three modifications to XP: the written documentation and requirements are to be managed by a tester, the planning game is to be modified so that it allows having multiple customer representatives and a requirements engineering phase in the beginning of a project to provide wider perspective of the product being developed. Boström et al. (S13) focus on the planning game as a way of extending XP practices. Nawrocki et al. (S47) have built a maturity model for XP (XPMM), a 4-level maturity model with a structure that resembles CMMI. Their aim was to build a maturity model that is simple and lightweight. Two papers focus on extending Scrum. Zanatta (S78) has developed an extension, called xScrum which consists of guidelines that allow Scrum to be compatible with the requirements management and requirements development process areas of CMMI. Marcal et al. (S38) suggest that few adaptations on Scrum, will make it much more compatible or even fully combinable with CMMI project management process areas. Manzoni and Price (S37) focus on extending RUP and state that an organization must customize its own practice and develop its own procedures to satisfy key practices of CMMI not supported by RUP. Zuser et al. (S79) propose a standard for quality support in software process models. Port and Bui (S59) introduce two strategies to obtain compatibility of the agile and disciplined approaches. One adding cost-benefit to the agile approaches, the other modulates development iteration size to maximize the expected cost-benefit for each iteration. Visconti and Cook (S72) have developed an ideal process model for agile approaches.

Laurila (S30) suggests the creation of a CMMI like framework that is more compatible with agile ideas. Instead of a plan-driven focus, the framework should embrace change. Stålhane and Hanssen (S64) suggest that the guidelines of ISO should include more specific guidelines, for example a definition of acceptable reviews.

Risk management is another approach that can assist the process of obtaining compatibility between the agile and the disciplined approaches and help organizations answer the question of how much documentation is enough (S7). Boehm and Turner (S9, S10, S11) therefore suggest a risk-based approach when incorporating both agile and disciplined practices and principles. They have developed a five step model. Galal-edeen (S19) discusses two approaches for obtaining compatibility, the ambidextrous organization approach and the risk-based approach. They found the risk-based approach of Boehm and Turner most useful, as the home grounds concept is practical when dealing with development projects. Geras et al. (S20) also concludes that the home grounds of Boehm and Turner represent a way to obtain compatibility between the agile and the disciplined approaches.

Proposition 2: Obtaining compatibility of the agile and the disciplined software development approaches requires an extension of either one or both approaches.

4.3 The challenges for obtaining compatibility

The analysis revealed several challenges when using an agile software development approach and at the same time being compliant with a quality assurance standard. The two main challenges identified in the literature are on the issues of handling the documentation and the requirements. This section deals with these two issues and the advice presented by the literature to overcome these.

4.3.1 The documentation

A difference between the disciplined approaches controlled by quality standards and the agile approaches is the amount of documentation required. Agile approaches do not support the degree of documentation demanded by disciplined approaches (S8, S53, S67, S78). The different focus on documentation is also reflected in the agile manifest which says “Working software over comprehensive documentation” [31]. In the case study presented by Kähkönen and Abrahamsson (S28) they had to do some additional documentation to comply with the standard. According to Nawrocki et al. (S45) the lack of documentation is the main weakness of XP. Fortunately both agile (S67) and disciplined (S3) approaches are highly flexible and adaptable. Some researchers state that documentation does not contradict the core principles of agility. Bos and Vriens (S12) argue that it is not necessary to write piles of documentation comply with CMM Level 2 and according to Diaz (S16) it is proven that the CMMI model can be applied in a light manner.

Research provides experiences and guidelines on how to deal with the amount of documentation. Stålhane and Hanssen (S64) suggest adding activities such as review meetings and writing design documents to the agile approaches, while still keeping the most agile ideas such as short iterations, building in increments and including the customer. The case study of Namioka and Bran (S43) showed how the short iterations and continuing updates to the documents enhanced the flexibility. Other guidelines are to treat the written documents as deliverables at the end of an iteration (S43), and to provide just enough documentation to help with enforcement of existing processes (S41). Process standards do not explicitly state how much documentation is required and many companies end up doing more documentation than needed.

Previous research also suggests extending the agile approaches to have a stronger emphasis on documentation. In parallel, disciplined approaches need to keep the documentation to a minimum (S54). The key to making the agile and the disciplined approaches compatible on the issue of documentation seems to be a happy medium between focusing on working software and on writing documents.

Proposition 3: To obtain compatibility between the agile and disciplined approaches the focus need to be on both working software and on documentation

4.3.2 The requirements

The agile method XP bases the requirements on user stories created by the customer [31], these differences in how the agile and the disciplined approaches handle requirements can cause problems in obtaining compatibility (S8). According to Nawrocki et. al. (S45) a weakness of XP is that this approach lacks documentation of requirements. User stories written in a plain business-like language cannot be used directly as requirements by a company wishing to comply with a process standard (S5, S77). Instead, each user story needs to be translated into functional test cases (S5). Nawrocki et al. (S45) propose a modification of the XP lifecycle introducing a requirements engineering phase at the beginning of a project and to make the tester responsible for managing the requirements.

Several tools supporting the XP process has been developed (S77). Melis (S42) suggests using such tools to manage the gathering of requirements and planning activities.

Proposition 4: The different strategies for handling requirements proposed by the agile and the disciplined approaches can cause problems when trying to obtain compatibility.

5 Conclusions

A large number of software companies have a desire to adopt an agile software development approach and at the same time comply with a quality standard. The two approaches seem contradicting but case studies which successfully balance agility and discipline is emerging. Therefore, the purpose of this paper is to determine whether the agile and the disciplined software development approaches are combinable or just compatible. This is done by reviewing previous research comparing the agile and discipline approaches for analysis or through a case study. The review includes 79 papers which were analysed using conventional content analysis and in this process coded using Atlas.ti V6.

The analysis showed that the agile and the disciplined approaches are highly compatible, but not combinable. In practice it is possible to implement several agile practices and principles in a software development process and at the same time be able to comply with a quality assurance standard, however the agility of a project and/or the practices and principles of an agile software development process will be harmed in some degree by the regulations of a disciplined quality standard or process model.

The agile approaches are mostly compared with CMMI level 2 or 3 or ISO. Few cases prove that it is possible to use parts of an agile approach with CMMI level 4 or 5. Agile practices and principles can also be introduced in the software development process when developing safety-critical software in order to make the development process more flexible. The research however also agrees that obtaining compatibility between the agile and the disciplined approaches is not straight forward. The research provides guidelines and different models on how to extend the agile approaches in order to make them fit the disciplined approaches. The amount of documentation and the way the requirements are handled seems to be the biggest difference between the agile and the disciplined approaches which proposes some challenges. Previous research does not distinguish between combinability and compatibility of the agile and the disciplined approaches, hence the researchers are advised to reconsider their terminology when comparing the agile and the disciplined approaches and adopt the definitions of compatible and combinable presented in this paper.

Only few empirical studies have been conducted on software development teams attempting to use an agile software development process in a disciplined setting. The research within this area would therefore gain from further empirical data on the compatibility of the two approaches and on the strengths, weaknesses and pitfalls when trying to obtain compatibility between the agile and the disciplined approaches.

References

1. Boehm, B. and Turner, R. (2003). Observations on balancing discipline and agility. In: Proceedings of the Agile Development Conference, IEEE Computer Society, Salt Lake City, Utah, USA, pp. 32-39.
2. Boehm, B. and Turner, R. (2004). Balancing agility and discipline: evaluating and integrating agile and plan-driven methods. In: Proceedings of the 26th International Conference on Software Engineering, IEEE Computer Society, Washington, DC, USA, pp. 718.
3. Dahlberg, H., Ruiz, F.S. and Olsson, C.M. (2006). THE ROLE OF EXTREME PROGRAMMING IN A PLAN-DRIVEN ORGANIZATION. In: The Transfer and Diffusion of Information Technology for Organizational Resilience: IFIP TC8 WG 8.6 International Working Conference, Springer, Galway, Ireland, pp. 291.

4. Nawrocki, J.R., Walter, B. and Wojciechowski, A. (2002). Comparison of CMM level 2 and eXtreme programming. Lecture notes in computer science, 288-297.
5. Turner, R. (2002). Agile Development: Good Process or Bad Attitude? Lecture notes in computer science, 134-144.
6. Turner, R. and Jain, A. (2002). Agile meets CMMI: Culture clash or common cause? Lecture notes in computer science, 153-165.
7. Boehm, B. and Turner, R. (2003). Using risk to balance agile and plan-driven methods. Computer, (36:6): 57-66.
8. Nawrocki, J., Olek, L., Jasinski, M., Paliswiat, B., Walter, B., Pietrzak, B. and Godek, P. (2006). Balancing agility and discipline with xprince. Lecture Notes in Computer Science, (3943:266).
9. Beck, K. and Andres, C. (2004). Extreme programming explained: embrace change, Addison-Wesley Professional, USA.
10. Pikkarainen, M. and Mäntyniemi, A. (2006). An approach for using CMMI in agile software development assessments: experiences from three case studies. In: SPICE 2006 conference, Luxemburg, pp. 4.
11. Marçal, A.S.C., de Freitas, B.C.C., Soares, F.S.F., Furtado, M.E.S., Maciel, T.M. and Belchior, A.D. (2008). Blending Scrum practices and CMMI project management process areas. Innovations in Systems and Software Engineering, (4:1): 17-29.
12. Hashmi, S.I. and Baik, J. (2007). Software Quality Assurance in XP and Spiral-A Comparative Study. In: International Conference on Computational Science and its Applications, 2007. ICCSA 2007. IEEE, Fukuoka, Japan, pp. 367.
13. Fritzsche, M. and Keil, P. (2007). Agile Methods and CMMI: Compatibility or Conflict? e-Informatica Software Engineering Journal, (1:1): 9-26.
14. Heeager, L.T. and Nielsen, P.A. (2009). Agile Software Development and its Compatibility with a Document-Driven Approach? A Case Study. In: Australasian Conference on Information Systems, Melbourne, Australien, pp. 205.
15. Theunissen, W.H., Kourie, D.G. and Watson, B.W. (2003). Standards and agile software development. In: Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology, South African Institute for Computer Scientists and Information Technologists, Republic of South Africa, Johannesburg, pp. 178.
16. Opelt, K. and Beeson, T. (2008). Agile Teams Require Agile QA: How to Make it Work, An Experience Report. In: Agile, 2008. AGILE'08. IEEE Computer Society, Toronto, pp. 229.
17. Reifer, D.J. (2003). XP and the CMM. IEEE Software, (20:3): 14-15.
18. Vinekar, V., Slinkman, C.W. and Nerur, S. (2006). Can agile and traditional systems development approaches coexist? An ambidextrous view. Information Systems Management, (23:3): 31-42.
19. Nerur, S., Mahapatra, R.K. and Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. Communications of the ACM, (48:5): 78.
20. Kähkönen, T. and Abrahamsson, P. (2004). Achieving CMMI level 2 with enhanced extreme programming approach. Lecture notes in computer science, (3009:378-392.
21. Nawrocki, J.R., Jasiñski, M., Walter, B. and Wojciechowski, A. (2002). Combining extreme programming with ISO 9000. Lecture Notes in Computer Science, 786-794.
22. Pikkarainen, M. (2009). Towards a Better Understanding of CMMI and Agile Integration-Multiple Case Study of Four Companies. In: Product-Focused Software Process Improvement: 10th International Conference, PROFES 2009, Springer, Oulu, Finland, pp. 401.
23. Rönkkö, M., Jarvi, A. and Makela, M.M. (2008). Measuring and Comparing the Adoption of Software Process Practices in the Software Product Industry. Lecture Notes in Computer Science, (5007:407-419.
24. Zanatta, A.L. and Vilain, P. (2006). Extending an Agile Method to Support Requirements Management and Development in Conformance to CMMI. HIFEN, (30:58): .
25. Webster, J. and Watson, R.T. (2002). Analyzing the past to prepare for the future: Writing a literature review. MIS Quarterly, (26:2): 13-23.

26. Hsieh, H.F. and Shannon, S.E. (2005). Three approaches to qualitative content analysis. Qualitative health research, (15:9): 1277-1288.
27. Muhr, T. (1991). ATLAS/ti—A prototype for the support of text interpretation. Qualitative Sociology, (14:4): 349-371.
28. Strauss, A.L. and Corbin, J. (1990). Basics of qualitative research: Grounded theory procedures and techniques, Sage Newbury Park, CA, .
29. Wehmeier, S. (2010), Oxford advanced learner's dictionary, [Homepage of Oxford University Press], [Online]. Available: http://www.oxfordadvancedlearnersdictionary.com/ [2010, December].
30. Turk, D., France, R. and Rumpe, B. (2002). Limitations of agile software processes. In: Third International Conference on eXtreme Programming and Agile Processes in Software Engineering, Citeseer, pp. 43.
31. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J. & Thomas, D. , (2001), Manifesto for Agile Software Development, . Available: http://agilemanifesto.org/ [2011, June, 30].

Appendix A. Studies included in the review

See http://heeager.com/files/combinability_or_compatibility_studies.pdf

Appendix B. Coding schemes

See http://heeager.com/files/combinability_or_compatibility_coding_schemes.pdf

Adopting Quality Assurance Technology in Customer–Vendor Relationships: A Case Study of How Interorganizational Relationships Influence the Process

Lise Tordrup Heeager and Gitte Tjørnehøj

Abstract Quality assurance technology is a formal control mechanism aiming at increasing the quality of the product exchanged between vendors and customers. Studies of the adoption of this technology in the field of system development rarely focus on the role of the relationship between the customer and vendor in the process. We have studied how the process of adopting quality assurance technology by a small Danish IT vendor developing pharmacy software for a customer in the public sector was influenced by the relationship with the customer. The case study showed that the adoption process was shaped to a high degree by the relationship and vice versa. The prior high level of trust and mutual knowledge helped the parties negotiate mutually feasible solutions throughout the adoption process. We thus advise enhancing trust-building processes to strengthen the relationships and to balance formal control and social control to increase the likelihood of a successful outcome of the adoption of quality assurance technology in a customer–vendor relationship.

Keywords Quality assurance · CMMI · ISO · GAMP · Interorganizational relationships · Trust · Control

1 Introduction

Quality assurance has its roots in Total Quality Management (TQM) [1] and is a formal behavioural control mechanism [2, 3] aiming at increasing the quality of an industrial or tailor-made product exchanged between vendors and customers. In the field of IT systems development, we find two main approaches to quality assurance and a row of combinations and adaptations of these. CMM(I) [4, 5] is the main approach to the improvement of software processes, while the ISO 9000

L.T. Heeager (✉)
Aalborg University, Aalborg, Danmark
e-mail: liseh@cs.aau.dk

series of standards [6] and QFD [7] are the main standards for quality improvement. BOOTSTRAP [8] is an example of an approach that builds on both the main principles.

Many IT vendors strive to become certified as compliant with the different standards and norms as they hope it will secure them a beneficial market position, while others just aim at improving their work processes and raising the quality of their products. Often, these companies initiate this effort as part of their long-term business strategy and not in response to the requirements of their customers.

The adoption of quality assurance can be studied either with a focus on adoption in the vendor organization or on the customer requirements and validation work or focussing on the role of the relationship between the two in the adoption process. Interestingly, CMM has norms both for vendors improving software processes and for customers improving their acquisitions process, but not for the mutual process [9]. Only little research has focussed on the relationship between customers and IT vendors when adopting quality assurance technology.

We have therefore investigated how the process of adopting quality assurance standards is influenced by the relationship between a customer and an IT vendor. We achieve this through a case study of the adoption of the GAMP standard [10] by a small Danish IT vendor (PharmSoft) developing pharmacy software for a customer in the public sector. This customer had geographically distributed user sites, each with separate management, but the cooperation with PharmSoft was organized and managed through a central department. The project management of PharmSoft worked closely with this department describing system requirements, while a cross-organizational steering committee prioritized the development resources. We study this case as the adoption of quality assurance technology in an interorganizational relationship between the vendor and the customer [11].

The chapter is structured as follows. First, we introduce the GAMP standard and the validation process for computer systems in Section 2. Then, we present the theory of interorganizational relationships, focussing on the concepts of trust and control [2, 3] and further attributes and processes [11] in Section 3. After describing our research approach (Section 4), we present the analysis of the adoption of quality assurance in PharmSoft (Section 5) followed by an analysis of the customer–vendor relationship of PharmSoft in Section 6. The interorganizational relationship and its influence on the adoption process are discussed in Section 7 and the conclusion is found in Section 8.

2 Quality Assurance in the Pharmaceutical Sector and the GAMP Standard

In Denmark, suppliers within the pharmaceutical sector are required to validate the quality of their delivery to the public sector. This includes IT systems dealing with the production and handling of medicine if they are part of such supplier systems. The GAMP4 Guide for Validation of Automated Systems launched in December 2001 by ISPE (International Society for Pharmaceutical Engineering) is widely used in Europe and was chosen as the standard by the customer in this case. GAMP4 is part of the GxP family of standards (Good Practice quality guidelines and regulations), which is used by the pharmaceutical and food industries and includes Good Manufacturing Practice (GMP), Good Laboratory Practice (GLP), Good Documentation Practice (GDP) and Good Clinical Practice (GCP) [10].

The central activity of GAMP is the validation of the automated system to ensure correct operation by a documented proof. To the extent that the automated system is an IT system, the system development process and its products should be validated too.

In comparison with the ISO 9001 standard, GAMP4 has stricter requirements for traceability from the customer requirements to the implemented system changes, for more detailed specification documentation and for standard operating procedures [12].

3 Interorganizational Relationships

At the beginning of this study, we focussed on the adoption process of the vendor but experienced that this picture did not reveal the whole situation. Thus, we have studied the adoption process of quality assurance as being joint between the customer and the vendor and thus being acted out within their relationship. We base our analysis on the theory of interorganizational relationships [2, 3] to understand this adoption process from both sides and its outcome.

The nature of an interorganizational relationship in general has a substantial influence on how successful the outcome of the partnership/cooperation is [2, 11, 13, 14] and how high the level of confidence in a partner cooperation is [2]. Both trust and control can contribute to building confidence in partner cooperation in interorganizational relationships. "A higher trust level does not automatically dictate a lowering of the control level and vice versa" [2].

Quality assurance technology is a formal behavioural control mechanism. Control mechanisms are "organizational arrangements designed to determine and influence what organization members will do" [2]. Formal control is an external measure-based control that "emphasizes the establishment and utilization of formal rules, procedures and policies to monitor and reward desirable performance" [3] in opposition to social control (or informal control or internal value-based control), which relies on shared norms, values, culture and internalization of goals as means to reach the desired outcome.

Trust plays an important role in achieving a successful relationship [11, 13–17]. Trust is the "positive expectations about another's motives with respect to oneself in situations entailing risk" [2]. Goodwill trust is trust in one's good faith, good intentions and integrity, whereas competence trust is trust in one's ability to do appropriate things, not trust in the other party's intention to do so. Having goodwill trust in a relationship, the partner will be less concerned with problems of cooperation [3]. It is noticeable that trust is hard to build, but easy to violate [15].

Both formal and social control can be successfully implemented in an interorganizational relationship, but may have different impacts on the level of trust. Formal control mechanisms can be a hindrance to a high level of trust, through their constraining of people's autonomy, whereas social control as a by-product can be trust-building as it often takes the form of socializing and interaction. There may be a contradistinction between trust and formal control mechanisms such as quality assurance when it comes to building confidence in a relationship.

Complete trust in a partner could point to no or a very low need for control mechanisms, but exclusive reliance on trust in relationships introduces an increased risk of failure, as structures or potential problems will be disregarded. Relying on excessive structural control can, however, hurt performance. Most projects therefore need a balance between trust and control [16].

To understand the influence of the nature of the relationship on the adoption process, we also include more parameters in our study of the relationship. We adopt the attributes and processes of Goles and Chin [11]. Through a thorough literature review of previous research, they constructed a list of factors playing a significant role in relationships and grouped these into attributes and processes. The five attributes that contribute to the functionality and harmony of the relationship are commitment, consensus, cultural compatibility, flexibility and interdependence and the five processes that develop the attributes are communication, conflict resolution, coordination, cooperation and integration [11]. Our literature study of papers on interorganizational relationships confirmed the conclusion that these attributes and processes are those most significant for a successful relationship [2, 3, 13–17].

Figure 1 is a framework illustrating the process of adopting quality assurance standards in an interorganizational relationship and the influencing factors. The adoption process takes place and is influenced by the attributes and processes of the relationship between the customer and the vendor. However, the context in which this relationship is acted out will also influence the adoption process. As time passes

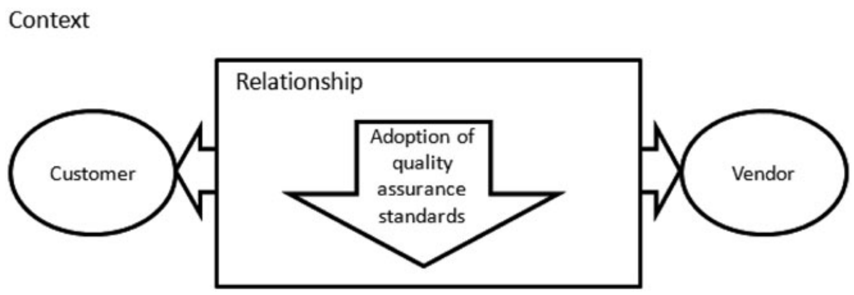


Fig. 1 Adopting quality assurance standards in an interorganizational relationship

and the adoption process develops, the relative importance of the influencing factors, e.g. the attributes and processes [11], changes.

4 The Research Approach

The case study was organized as an interpretive single case study [18] and ran from September 2007 until June 2008, just before and after the first external audit of the vendor (see Fig. 2). Our research interest was in understanding the process of adopting quality assurance.

The data were collected and analysed in two phases. This gave us a rich history on which to base our data collection and analysis. In the first phase, we studied the adoption of the GAMP standard from the view of the vendor and focussed on the internal processes. The interviews included employees from PharmSoft: the project manager and three developers. They were semi-structured and organized as diagnostic interviews [19]. The purpose was to gain an initial understanding of the adoption process of the vendor and to identify the employees' opinions and views on the situation. Furthermore, one researcher made observations of the vendor for 4 months, building personal knowledge of the case. This phase showed us that studying this process only from the vendor's side would not reveal a full picture of the situation. Thus, we chose to involve the customer and focus on the interorganizational relationship.

In the second phase, we shifted the level of focus and studied the adoption process from an interorganizational perspective. The phase included five qualitative interviews with the project manager from PharmSoft, a developer from PharmSoft, two users of the system and a customer representative. These interviews were semi-structured and explicitly seeking the interviewee's personal view of the situation.

During the two phases, all the interviews were recorded and transcribed and documents relevant to the adoption process from both the vendor and the mutual project were collected.

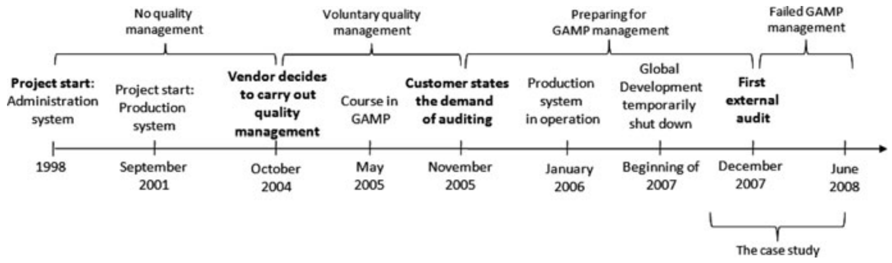

No quality management
Voluntary quality management
Preparing for GAMP management
Failed GAMP management
Project start: Administration system
Project start: Production system
Vendor decides to carry out quality management
Course in GAMP
Customer states the demand of auditing
Production system in operation
Global Development temporarily shut down
First external audit
1998
September 2001
October 2004
May 2005
November 2005
January 2006
Beginning of 2007
December 2007
June 2008
The case study

Fig. 2 Timeline

We analysed the data iteratively through two analyses both in between the phases and after the second phase. First, we compiled a historical map (see Fig. 2) of important events in the adoption process by working through the data and discussing them with the project manager of the IT vendor. We identified four phases: no quality management, voluntary quality management, preparing for GAMP management and failed GAMP management. Based on this, we wrote up the case story as seen in Section 5. Then, we analysed the customer–vendor relationship, according to the attributes and processes of Goles and Chin [11], with special emphasis on trust and control [2, 3]. We investigated each interviewee's perception of the relationship to clarify whether there were different perceptions between the customer and vendor and to clarify any internal differences. The findings were written up and presented to the interviewees. To our surprise, the perception of the relationship was shared overall between the parties; thus, we have reported the results in Section 6 as one description, though pointing out the differences.

The interpretation of this case emerged during the rather elaborate analysis process and mainly builds on the concepts of trust and control [3].

5 Adoption of Quality Assurance at PharmSoft

The adoption process had the form of an iterative long-term and somewhat experimental adoption process resulting from ongoing negotiations between the two parties of the customer–vendor relationship.

The system development process was initiated in 1998. The first subsystems developed did not require validation of the software. In September 2001, the planning of a production subsystem was initiated, and during the next 2 years, the parties realized that this system would be the target of validation and they discussed the subject initially and briefly. However, no concrete requirements for the validation process were made and no quality assurance standards were decided on, since the customer organization was in doubt regarding how to handle the legal requirements.

In light of this, PharmSoft decided in October 2004 to initiate a quality assurance effort on its own. Inspired by traditional quality assurance practice in the IT industry, the employees described their existing working practices in formal procedures to document their initial quality level. The main focus of this work was system documentation, document management, requirements and change management. They organized the adoption process with the person responsible for the quality assurance, writing up initial procedures that were reviewed, discussed and even changed in workshops with the employees. The customer organization was of course informed about this work and the results.

The same autumn, at a steering committee meeting, a powerful manager of the customer organization presented and advocated the GAMP4 standard as an appropriate choice for organizing the quality assurance effort. This sparked a clarification in the customer organization on how to meet the public requirements and after some time the choice of the GAMP standard was approved. Both the customer and

PharmSoft made an effort to learn and understand the GAMP standard, including a joint course on GAMP in May 2005. Eventually, only 2 months before the production system was implemented in November 2005, the customer organization made an official demand for a formal audit according to the GAMP standard. Neither the customer organization nor PharmSoft, however, knew what that actually meant, since both lacked experience in interpreting a quality standard and design or even implementing procedures according to it, even though both had responsibilities in the validation work according to the standard.

PharmSoft prepared well for the audit, shaping up the existing quality assurance system and striving to implement procedures to meet the rest of the standard requirements to do as well as possible. The customer organization, on the other hand, did not expect PharmSoft to pass the audit at the first attempt. They had called for it as part of the clarification of how to implement GAMP in the project.

PharmSoft failed the audit by external and independent auditors, with remarks mostly on the quality assurance organization, test and test documentation, document management requiring much more printed and signed documentation (contrary to the existing online documentation) and requirement specification. Both test and requirement specifications are shared working practices between the customer organization and PharmSoft. An example is how the customer and PharmSoft collaborated and negotiated the design of the new test procedure: the test practice until December 2007 had been developed over years by the parties until they agreed on having reached a sufficiently high level of quality of the system implemented in each delivery. However, since GAMP had stricter requirements for documentation and for the separation of validation and software development, a new test practice was launched in spring 2008. The first test conducted thereafter did not work as expected. Too many flaws were found too late in the test period. As a result, the customer and PharmSoft decided to integrate the new procedure with the old test practice into yet another new test procedure for the system.

After the audit, both parties decided to buy in external experts on quality assurance to help them reach an appropriate design and implementation of an appropriate quality assurance system complying with GAMP4, but highly adapted to existing practice in the project. As of November 2008, PharmSoft had not yet passed the certification, but the adoption process was still progressing to both parties' satisfaction.

6 Analysis of the Interorganizational Relationship of PharmSoft

The relationship is analysed according to the five attributes and processes (Tables 1 and 2). The customer and PharmSoft describe the level of trust between them as high. Their communication and cooperation is characterized by willingness, honesty and flexibility. By these means, they resolve conflicts peacefully, preventing problems from growing too big. Through the years, they have built a high mutual knowledge and come to consensus in many fields. The customer is

Table 1 Summary of the findings from the attributes analysis of the case

Attributes	
Interdependence	PharmSoft is highly dependent on the contract with the customer organization, since it is its main customer. However, since the system has become very complex and difficult to maintain over the years, both the project manager and the developers from PharmSoft perceive this dependency to be bilateral. The less trusting parts of the customer organization have, however, been giving thought to looking for a substitute vendor to keep down the expenses and raise the quality of the system
Flexibility	The main responsible person from the customer and the system users perceive PharmSoft as flexible, when circumstances change. Limited resources (the employees) combined with the complexities of the tasks, however, restrict this flexibility. The customer representative accepts this as reasonable
Cultural compatibility	The cultural compatibility is high since both PharmSoft and the customer are experienced actors on the Danish IT market and thus overall share this business culture, knowing the basic legal requirements and formal and informal rules of practice in the market
Commitment	PharmSoft is very committed to the relationship, according to the customer representatives. It does all it can to please the users. The representative emphasized the willingness of PharmSoft to discuss problems and find solutions On the other hand, commitment from the customer towards the relationship is diverse. The parts of the organization working closely with PharmSoft are highly committed but other parts show some resistance, as mentioned under the other attributes
Consensus	The relationship is characterized by a remarkable consensus between the parties The customer and PharmSoft express an overall unity built through a mutual understanding and close collaboration over time. Their frequent communication and dialogues help build this consensus Within the customer organization, the conflict of interest is evident. The central department and close vendor contact prioritize a swift adoption of GAMP, while the user sites prioritize the development of new and added functionality. In between, PharmSoft finds it difficult to know whom to serve first

Table 2 Summary of the findings from the attribute-building-process analysis of the case

Processes	
Cooperation	PharmSoft and the customer describe their cooperation as close and well functioning. The vendor often successfully pleases the users and the vendor feels understood by the customer
Communication	The communication between PharmSoft and the customer representative is frequent. They communicate to manage the development, to confirm requirement specifications and whenever an issue has to be discussed. Both parties perceive their communication as honest

Table 2 (continued)

Processes	
	PharmSoft communicates with the system users at the steering committee once a month and when local system adaptations or add-ons need to be negotiated. This latter communication is either by phone or email. Both PharmSoft and the customer prefer communicating by phone as they find that misunderstandings occur more often in written form
Conflict resolution	Both the customer and PharmSoft said that they have no conflicts, only small disagreements that are solved through communication
Coordination	Coordination was not an important process of this case
Integration	Integration was not an important process of this case
Creating mutual knowledge	The mutual knowledge is highest between PharmSoft and the customer representative and between PharmSoft and the system users. This knowledge has been built through time, mainly at the monthly steering committee meetings that involve a stable group of people. Mutual understanding was mentioned by both PharmSoft and the customer as being important for a successful relationship

furthermore satisfied overall with the system delivered by PharmSoft. We will therefore characterize this as a trusting relationship. However, some parts of the customer organization are less satisfied with the whole arrangement, which puts the relationship under some kind of pressure.

7 Discussion

The relationship between the customer and PharmSoft is characterized overall by both mutual goodwill trust and competence trust as both parties expressed high trust in the other party's good intention, commitment to the cooperation and ability to cope with the tasks. The customer expressed the belief that PharmSoft does everything possible to please them, and the further belief that PharmSoft will achieve the validation within an acceptable time limit. PharmSoft accepted the quality assurance management and appreciated the fact that the customer negotiated the requirements of the standard.

The high level of trust has been built through years of successful, close cooperation involving a lot of direct communication between a stable group of people including the project management of PharmSoft and the customer representative and several of the system users.

The relationship between the customer and PharmSoft was mostly based on social control and high mutual trust already before starting to develop the production system. The choice of quality assurance standard and thereby the official beginning of quality assurance was diminished due to the trusting relationship. Neither the customer nor PharmSoft experienced a need for further control. The need for validation arose solely due to the legal requirement.

The voluntary quality management was triggered when PharmSoft realized that the subsystem would be subject to the legal requirement for validation. To meet a demand from its customer, it initiated a quality management effort focussing on immediate beneficial improvements in its work practice. This worked as a sign of flexibility and willingness to cooperate from PharmSoft.

The next months showed that the adoption of quality assurance technology was perceived as a mutual task by the parties and that they cooperated in order to ease the process. We find evidence that the driver of this was the trusting relationship as shown by the fact that PharmSoft decided to initiate the quality management on its own, in the joint course on GAMP and in the way it negotiated the requirements for the quality management.

The process that led to the decision on GAMP was sparked by a manager at the customer, who as an outsider of the trusting relationship pushed the model and the need for orderly validation on a steering committee meeting. We do not know how and when the decision was made, but it ran seamlessly into the third phase: preparing for GAMP quality management. The fact that it took an outsider to make the official decision on quality assurance shows that trust can diminish the felt need for the implementation of a formal control mechanism in a well-functioning relationship. Subsequently, a period of preparing for the upcoming audit began. Even though PharmSoft made an effort to adopt the quality management of GAMP, it was well known prior to the audit on both sides that it was likely to fail.

As expected, PharmSoft failed the audit and, as the problems now were explicitly known, the customer and PharmSoft were forced to deal with the situation. However, they managed to overcome this situation by cooperating and found a mutual, satisfactory solution, e.g. this was shown in the way they handled the negotiation of the test procedures (see Section 5).

The case of PharmSoft showed that an interorganizational relationship may influence the process of quality assurance adoption in several ways. The level of trust can impact on the felt need for more formal control to an extent where both of the participants are ready to rely solely on trust and, as shown in this case, trust between parties can diminish the formal decision of quality assurance adoption. However, trust between parties in an interorganizational relationship is very helpful when adopting quality assurance. Trust makes the parties perceive the adoption process as a mutual task and cooperate to smooth the adoption process for both parties, reducing the number of problems owing to conflicts and helping in the process of resolving the conflicts that do arise. Due to few conflicts, the quality and time of delivery of the outcome will not be pressured. If the adoption process is successful, this mutual process will enhance the trust level even further.

8 Conclusion

In this chapter, we investigate how the process of adopting quality assurance standards is influenced by the relationship between a customer and an IT vendor. This was achieved through an interpretive case study of an IT vendor striving to adopt GAMP in a relationship with its customer.

From this case, we learn that a successful trusting relationship between an IT vendor and a customer can be threatened by the adoption of the formal control mechanism quality assurance. However, we also found that the trusting and close relationship helped the parties to overcome this threat and integrate the quality assurance technology into their relationship.

Our research shows that not only the parties and their internal capacities matter in the adoption of quality assurance technology, but that the relationship between them and the surrounding context also influence the process itself and the outcome. We have presented a framework illustrating the adoption of quality assurance in an interorganizational relationship based on the theory of interorganizational relationships [11], which we found useful in the analysis of the case. We suggest that further studies could adopt this broad scope for understanding the adoption of quality assurance technology.

Even though it was not our purpose, we found indications in our study that adopting quality assurance to a large extent does influence the customer–vendor relationship and that especially relationships that are based on trust can be harmed. This may be a somewhat neglected consequence of adopting quality assurance technology that deserves more attention in future research.

References

1. Deming, W. E. (1982). *Out of the Crisis*. Cambridge, MA: Productivity Press.
2. Das, T. K., and Teng, B.-S. (1998). Between trust and control: Developing confidence in partner cooperation in alliances. *Academy of Management* **23**, 491–512.
3. Das, T. K., and Teng, B.-S. (2001). Trust, control, and risk in strategic alliances: An integrated framework. *Organization Studies* **22**, 251–283.
4. CMMI Product Team. (2001). *Capability Maturity Model Integration*. Pittsbourgh, PA: Carnagie Mellon Software Engineering Institute.
5. Paulk, M. C., Chrissis, M. B., and Weber, C. (1993). *Capability Maturity Model for Software Version 1.1*. Pittsburgh, PA: Software Engineering Institute.
6. Hoyle, D. (2005). *ISO 9000 Quality Systems Handbook*. Boston: Elsevier Science and Technology.
7. Akao, Y., and Mazur, G. H. (2003). The leading edge in QFD: Past, present and future. *International Journal of Quality & Reliability Management* **20**, 20–35.
8. Kuvaja, P., and Bicego, A. (1994). BOOTSTRAP – a European assessment methodology. *Software Quality Journal* **3**, 117–127.
9. Bjerknes, G., and Mathiassen, L. (2000). Improving the Customer-Supplier Relation in IT Development. *Proceedings of the 33rd Hawaii International Conference on System Sciences*, 1–10.
10. ISPE. (2008, September 29). *International Society for Pharmaceutical Engineering*. Retrieved from http://www.ispe.org.
11. Goles, T., and Chin, W. W. (2005). Information systems outsourcing relationship factors: Detailed conceptualization and initial evidence. *The DATABASE for Advances in Information Systems* **36**, 47–67.
12. Wright, G. (2003). *Achieving ISO 9001 Certification for an XP Company*. Berlin, Heidelberg: Springer, 43–50.

13. Holmström, H., Conchúir, E. Ó., Ågerfalk, P. J., and Fitzgerald, B. (2006). The Irish Bridge: A case study of the dual role in offshore sourcing relationships. *Twenty-Seventh International Conference on Information Systems*, 513–526.
14. Kern, T. (1997). The gestalt of an information technology outsourcing relationship: An exploratory analysis. *International Conference on Information Systems*, 37–58.
15. Brunard, V., and Kleiner, B. H. (1994). Developing trustful and co-operative relationships. *Leadership & Organization Development Journal* **15**, 3–5.
16. Sabherwal, R. (1999). The role of trust in outsourced IS development. *Communications of the ACM* **42**, 80–86.
17. Zaheer, A., McEvily, B., and Perrone, V. (1998). Does trust matter? Exploring the effects of interorganizational and interpersonal trust on performance. *Organization Science* **9**, 141–159.
18. Walsham, G. (1995). Interpretive case studies in IS research: Nature and method. *Operational Research Society Ltd*, 74–81.
19. Iversen, J., Nielsen, P. A., and Nørbjerg, J. (1999). Problem diagnosis software process improvement. *The DATABASE for Advances in Information Systems* **30**, 66–81.

12-1-2009

Agile Software Development and its Compatibility with a Document-Driven Approach? A Case Study

Lise Tordrup Heeager
Department of Computer Science, Aalborg University, liseh@cs.aau.dk

Peter Axel Nielsen
Department of Computer Science, Aalborg University, pan@cs.aau.dk

Agile Software Development and its Compatibility with a Document-Driven Approach? A Case Study

Lise Tordrup Heeager & Peter Axel Nielsen
Department of Computer Science
Aalborg University
Denmark
Email: liseh@cs.aau.dk, pan@cs.aau.dk

Abstract

It is generally assumed among software developers and managers that a document-driven development process is incompatible with an agile development process. There are few reports on this issue and even less empirical research documenting the assumed incompatibility. More and more software companies however have a desire to adopt agile development processes while maintaining compliance with a quality assurance standard or even a process standard. We have studied the software development process of a Danish pharmaceutical company. For market reasons the company has to comply the US Food and Drug Administration's standards for software development. The company has also successfully implemented significant parts of the agile methodology Scrum. We describe this case and we analyse using Soft Systems Methodology how well the development process combines these diverging sets of process requirements. We find that despite much effort the agile development process suffers from its adaptation to the FDA standard, but at the same time many of the genuine qualities of an agile process have been maintained. From this case study we discuss the implications for development companies facing the challenge of implementing a mixed development approach.

Keywords: Agile software development, document-driven software development, process standards, FDA.

INTRODUCTION

Embedded software is a growing industry and a significant part of embedded software is safety critical (e.g., Knight 2002). Whether this safety-critical embedded software is used to control potentially dangerous devices, machinery or infrastructures there is very often a set of requirements compiled into standards, which govern or should govern how software is developed. The European Space Agency has a number of such process standards (Jones et al. 2002). The CMMI (e.g., Chrissis et al. 2003) was invented and promoted for the US Department of Defence as a maturity and process standard for all its software subcontractors to be compliant with. The ISO9000 family of standards for quality management has been suggested and used to standardize also how software companies manage their quality assurance when developing software (Mutafelija & Stromberg 2003; McMichael & Lombardi 2007). For clinical software and embedded software in clinical devices there are several process standards. A medical device for the US market has to be approved by the FDA (Food and Drug Administration) and as part of this approval the software development processes have to be compliant with their process standards (FDA 2002). Medical device approval is becoming increasingly expensive and the desire to be compliant from an early stage with FDA's process standards is thus increasing (Lee et al. 2006; Abdeen et al. 2007). There are many reports suggesting that these process requirements are indeed necessary as safety-critical software would otherwise be too error-prone and too risky (Rakitin 2006; Schrenker 2006). These process standards all require that requirements are specified prior to their design and implementation, that documentation is a cornerstone to achieve high quality, and that documents are only changed through controlled procedures, i.e., what is called a document-driven approach to systems development.

Agile development, on the other hand, is based on the assumptions that programs and programming is primary – not documentation. The agile manifesto for software development has this as one of four major principles. Agile development seeks to increase software quality by adhering to a very flexible process (e.g., Cockburn 2002; Beck & Andres 2004) that relies on the competence of the software developers and not on defined processes (Aaen 2003). Agile development is increasingly popular among software companies as it provides a better explanation of how requirements change over time and how the resulting software therefore fits the final requirements.

There has already been a discussion of whether these two sets of assumptions, agile development and document-driven development, can co-exist and even be combined into a mixed approach. A notable debate has been between Beck and Boehm on whether an agile approach is paradigmatically different from a plan-driven approach (Beck & Boehm 2003). Some conclude from a theoretical study of the approaches that they are very alike and can co-exist (Turner & Jain 2002; Theunissen et al. 2003; Fritzsche & Keil 2007). Experience reports on a mixed approach are few, but some report how they have taken a rigorous process standard like CMMI as the primary framework and apply the agile approach Scrum within that (Sutherland et al. 2007; Jakobsen & Johnson 2008). Others report from experience with adding quality assurance processes to an agile development approach (Opelt & Beeson 2008). (Rottier & Rodrigues 2008) report from their experience with introducing Scrum in Cochlear, a medical device company. The research on these issues has so far been either theoretical (and also detached from real-world experience) or it has been mere reporting on experience (without any connection to theory). We want with this paper to contribute to a better understanding of how a mixed approach may be practiced and how well it is practiced.

In the following section we explain how we have performed our research as an interpretive case study. This is followed by a section where we describe the case we have studied. Then we present an analysis of the case, together with an evaluation of the mixed approach in practice. After this follows a section where we discuss the findings and then finally we conclude the paper.

THE RESEARCH APPROACH

We studied a pharmaceutical company in Denmark. In particular we studied how a clinical product with embedded software is developed. The product development involves pharmaceutical and clinical researchers responsible for the drugs, process engineers responsible for production facilities, mechanical engineers responsible for the products mechanical functions dispensing the drugs, embedded hardware engineers responsible for the computer chips and communication controlling the mechanics, and software developers responsible for the software controlling the hardware. The project is large, involving more than a hundred engineers and developers and lasting several years. Within this project we have specifically studied the software team and how they have implemented a software development approach that is a mix of a document-driven FDA-compliant process and an agile process.

The study was organized as an single case study. Our research interest was in understanding in an interpretive way (Walsham 1995) how they develop software with this mixed approach and how well it supported their intentions of a reasonably effective and efficient development practice. In particular we wanted to understand this in detail and in the outset we did not know exactly which details we were looking for. We also wanted to understand the developers different views on the mixed approach. For these reasons an interpretive case study approach is appropriate.

Phase	Data collection	Data documentation	Data analysis
1	7 qualitative interviews based on an interview guide	Interviews audio recorded and transcribed	Using Soft Systems Methodology to explore the problem situation
2	8 (4+4) qualitative interviews based on 2 interview guides	Interviews audio recorded and transcribed	Using Soft Systems Methodology with a particular focus on the mixed approach
3	Seminar presentation and validation with 11 members of the software team	Meeting audio recorded and transcribed	Validation of findings about the mixed approach

Table 1: Phases in the interpretive research approach

We collected and analysed the empirical data iteratively through three phases, see Table 1. The first phase was based on a semi-structured interview guide that included the subjects: FDA requirements, agile software development, the requirement specification, handling of errors and general strengths and weaknesses. Seven interviews were conducted with: the project manager, the software architect, a software tester and four developers. The interviews were recorded and transcribed. The analysis was performed using Soft Systems Methodology (SMM) (Checkland & Scholes 1990) by (i) drawing rich pictures of the situation and problems encountered; (ii) creating root definitions and conceptual models of activity systems; and (iii) a comparison between the models and the interviews.

The second phase included eight interviews with: the project manager, the software architect, a tester and five developers. Four of these interviewees participated in the first phase, whereas the last four participated for the first time. The interviews were again audio recorded and transcribed. This second round of analysis also followed SSM, but now with a single human activity systems and its conceptual model. This model was used first to identify sub activities and then further to compare the intended mixed approach with the development practice of the software team.

In the third phase the findings from the second analysis were written into a report and presented to the software team in a seminar. The seminar included a presentation of the results and a discussion of these. The interviewees from the previous two phases participated.

CASE DESCRIPTION

As outlined in the previous section this is a case study of a pharmaceutical company currently developing a clinical device for dispensing drugs and information processing. The controlling of the drug dispensing is performed by embedded software. In total over 100 managers, researchers, engineers and developers are working on the device project. The project has been running for several years and is soon to enter the final stage of refining the product.

The software development team consists of 17 developers and their project manager working in close cooperation with the engineers developing the hardware. The software development team is divided into two sub teams; one sub team develops software for the remote control to device while the other sub team develops software for the part that actually controls the drug dispensing. Furthermore, two testers, a software architect and a project manager are assigned to the software project. Some years ago it was decided by the project manager that the software team should utilise parts of an agile methodology in their daily practice by introducing iterations of four-week periods each supposed to produce a tested increment of software. The software team chose Scrum (Schwaber & Beedle 2001) as their primary agile method. The overall device project still has to be compliant with a large number of standards, some which apply directly to the software development process (FDA 2002). Maintaining adherence to an agile approach and the FDA standards at the same time has been a difficult management task for the software team. The software project manager has subsequently formulated a mixed software process to solve this.

researchers have developed the model in Figure 1 based on the first round of interviews as a model of the mixed development approach. SSM (Checkland & Scholes 1990) was applied to the problem situation after the first round of interviewing and several human activity systems with accompanying root definitions and conceptual models were tried and then compared with the interview data. The status of the conceptual model in Figure 1 is that it expresses a consolidated view of the mixed approach. It is consolidated only to the extent that the interviewees agree that it expresses well the intentions and assumptions behind the mixed approach.

The input to the mixed development approach is a set of requirements stated at the higher levels of the project organization in overall requirements specifications. The requirements are placed in the project documents such as the product specification and human-computer interaction specification. These documents are developed outside of the software team.

The project is committed to a strict document structure with formal handovers between phases and between different sub projects. Changing the requirements in the product specification is extremely difficult and time-consuming because of the many stakeholders involved in approvals and handovers. The software requirement specification (SRS) is based on the overall specifications.

The SRS serves as the software backlog and is written by the software architect. The SRS constitute the foundation on which the developers must understand and develop their current development tasks (Activity 1). A hazard analysis enlightens the risks associated with the device and its operation, which has to be considered and mitigated. In order to develop a safe and user friendly device the software developers are required to have an understanding of the context of the device, e.g., patients' behaviour and physical space.

The developers will plan their iterations, i.e., iterating activities (Activity 2). Due to the size of the software team they are divided into sub teams during this process. The tasks will be written on post-it notes on a large board on the wall in the team's open office space in order to visually display: which tasks are to be solved, which currently are being solved and which have been solved.

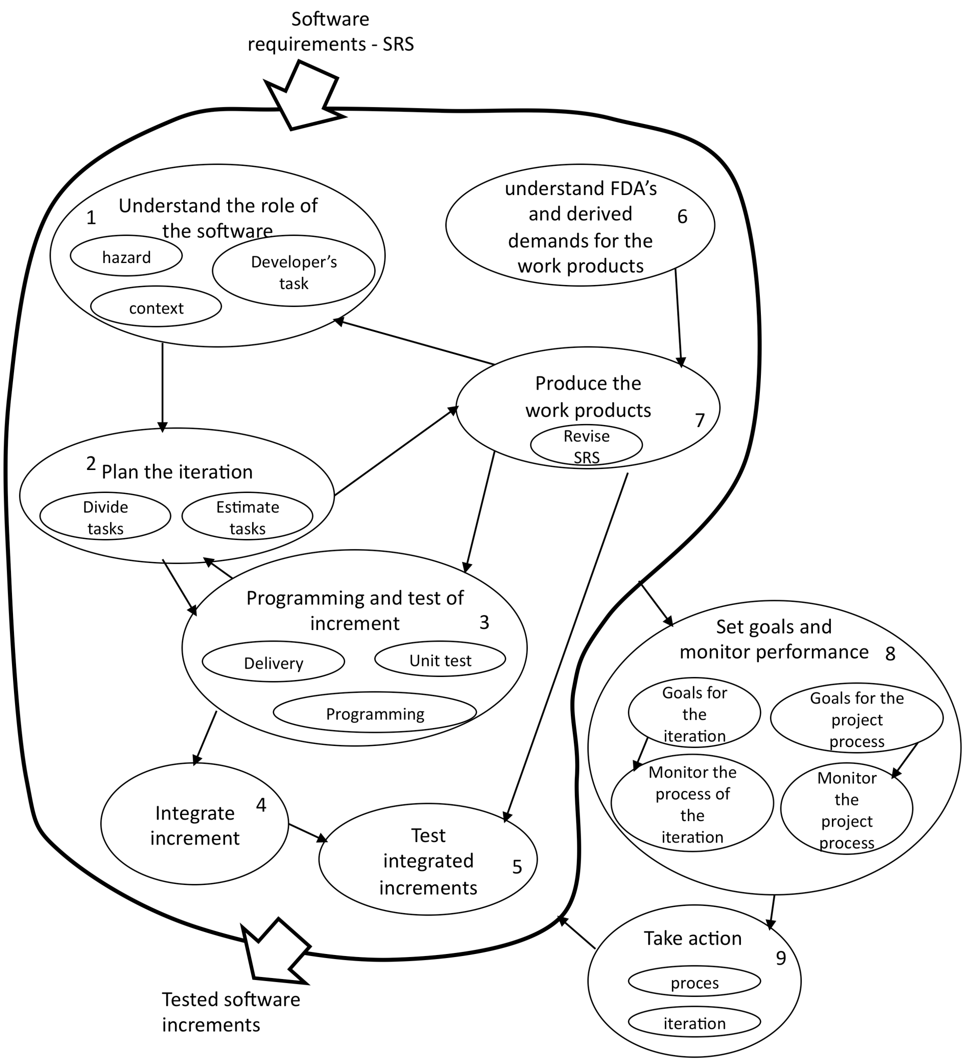

Software requirements - SRS
1 Understand the role of the software
hazard
Developer's task
context
understand FDA's and derived demands for the work products 6
Produce the work products 7
Revise SRS
2 Plan the iteration
Divide tasks
Estimate tasks
Programming and test of increment 3
Delivery
Unit test
Programming
Set goals and monitor performance 8
Goals for the iteration
Goals for the project process
Monitor the process of the iteration
Monitor the project process
Integrate increment 4
Test integrated increments 5
Take action 9
proces
iteration
Tested software increments

Figure 1: The software team's mixed development approach

The iterations contain programming, delivery and unit testing of the increments (Activity 3). The developers choose tasks from the board and then implement this piece of the software. All tasks are to be peer reviewed by a colleague before being delivered. The coordination during an iteration is done at stand-up meetings every morning lasting 15 minutes. The purpose of the stand-up meeting is for the developers to state their past, present and future (next 24 hours) working tasks in order to give the group an overview of what everybody is working on and make sure no one is getting stuck in a problem. The developers create the unit test cases when programming a task. The developers decide whether they write the test cases before or after programming. The unit tests are run automatically every 24 hours to ensure that new software has not implemented a new error. The software team's development manual states that they must have one hundred percent condition-decision coverage on unit tests.

The new increment, i.e., the result of an iteration, is continuously integrated with the existing system (Activity 4). After integration test (Activity 5) the increment is moved to the system engineering group which runs a device test. Errors are reported back to the software team trough an error reporting system. When an error is reported by the system engineering group several formal procedures must be followed.

The developers must understand the demands from FDA as well as the derived demands for the work products in order to produce these (Activity 6). The SRS is the most important work document in the software development team (Activity 7) and that will be gradually revised and detailed as the project progresses.

The development activities are monitored relative to the goals set for each iteration and for the entire project process (Activity 8). Progress for each iteration will be visualized in burn-down charts next to the task board on the wall in the open office space. Each iteration ends with an evaluation, which is an extended stand-up meeting where both the process and achieved increments are on the assessed. When needed action is taken in order to improve practice (Activity 9).

ANALYSIS OF THE DEVELOPMENT PROCESS

This section is an analysis of the current software development practice used by the software team. In effect it is an evaluation of the software team's practice based on the human activity system in Figure 1. The analysis consists of three parts. First, the overall structure of the development process is assessed; second, the activities of the development process are examined (including an analysis of how well the activity concerned is performed in the software team); third, the perspective will be broadened and the focus will be on the impact of the project structure, which the software team is a part of.

The Overall Structure of the Mixed Development Process

The mixed approach consists of operational activities that are performed either as sequential or as iterative, and of management activities that are both sequentially and iteratively performed, see Table 2. The core of the mixed process is primarily circular in the tight interchange between the activities: '2: Plan the iteration' and '3: Programming and test of increment', see Figure 1. Several other activities are undertaken concurrently, e.g., '7: produce the work products'; but these are largely sequential.

The inputs to the software development process are the overall requirements for the device as a whole as stated by the management of the device project – outside the software team. The requirements were frozen and written into different documents such as the device specification and the hazard analysis. The requirements were to a high degree formed at the beginning of the device project and were partly based on a user survey. The software team therefore has no direct customer contact or any contact with a potential end-user as recommended by agile methodologies. The software architect of the software team is responsible of interpreting the requirements in the project documents, write them up as software tasks and include them in the SRS. The SRS serves as the software backlog; however tasks are neither created nor prioritized by customers (neither in the sense of potential patients nor in the sense of the device project as a customer) as usually required in an agile approach. The software project's context has thus required sequential problem solving where the SRS could be detailed further, but overall requirements could not be changed.

The main activity of the development process is '3: Programming and test of increment'. This activity is strongly tied to the activity '2: Plan the iteration'. These are done iteratively and follow the Scrum methodology. The sprint backlog is created at the planning meeting and the tasks within it are estimated, the sprints are of four weeks durations and these are controlled by daily stand-up meetings.

The iterative activities are time-boxed and the output is supposed to be a fully working software increment. However in practice not all iterations deliver a new slice of the product and the increments are often not fully tested at the end of the iteration. The activities 4 and 5 were supposed to be iterative, however they are not. It was similarly the intention with activity 7 that it should be performed prior or at least in the beginning of every iteration, but that never happened.

Activities and their linkage	Sequential	Iterative
Operational activities	1, 4, 5, 6, 7	2, 3
Management activities	8, 9	8, 9

Table 2: The actual performance of the activities in the mixed approach

The dual nature of the management activities 8 and 9 reflects that there is in practice management of both the iterations and of the sequential activities.

The Software Team's Activities in the Development Process

In the following each of the activities of the development process of the software team will be analysed.

Understand the role of the software in the device (1). The hazard analysis is necessary due to the fact that the system is highly safety critical. The software team receives the results of the hazard analysis through documentation. Most of these results are formed as requirements for the software in order to mitigate the patients' exposure to the device's risks. To understand each development task the developers consult the SRS to get acquainted with the requirements for the relevant part of the system. If this information is not sufficient they contact other sources in the project, who have more information on this issue. The software team has in general a superficial understanding of the context in which the product is to be used. In the beginning of the project the developers participated in a course on general medicine. However, the knowledge gained through this course has not been maintained. The project has been ongoing for several years and some substitution and addition of developers has occured. The superficial understanding of the context is also caused by the non-existing user contact.

Plan the iteration (2) and Programming and test of increment (3). These activities are central to the development. The software team performs time-boxed development. Following Scrum they chose iterations of four weeks length. However, they recently expanded the length of an iteration to five weeks. The sprint backlog is created from the software backlog by the developers. They define the tasks for an iteration on the basis of the prioritization of the requirements and each task is then estimated. The tasks in an iteration are displayed on a large board in the open office alongside the burn-down chart displaying the progress in the iteration. Estimates often do not fit within the iteration and tasks are being postponed to the next iteration. This is a problem for two reasons: (1) the tasks which are postponed are often the same, and (2) the developers have gotten used to the fact that they never reach their stated goal for an iteration and have stopped striving for it. The attitude has become: "if we do not manage to implement it in this iteration, we will just do it in the next". The estimates slip not only due to lack of ability to judge the size of the tasks but also because of interference from other groups in the project. These interferences stem primarily from errors discovered and reported by the system engineering group during system test. Some of these are so severe that they required to be corrected right away – hence the time-box is broken. The hardware group and the human-computer interaction group also interfere inside an iteration with new requirements details just invented – hence without a discipline the time-box is again broken.

The iteration is controlled by the daily stand-up meetings (Activity 8). The duration of these is approximately 15 minutes. Due to the size of the software team the stand-up meetings have taken too long time when everybody gets to speak. Therefore one person from each sub team is now responsible of getting a short summary from his co-workers and presenting this at the stand-up meeting. This has however reduced the quality of the information presented. As one sub team is much bigger than the other sub team most of the meeting time concerns their tasks and is often to a high degree irrelevant to the other sub team.

All development tasks are to end with a peer review of developed software performed by a colleague before the task can end. Each developer is responsible for implementing this. But even though the developers participating in the interviews of this study expressed great satisfaction with the review process, too few peer reviews are actually conducted. This may have several reasons: (1) the developers perceive the implementation of additional software more important than doing the peer review, (2) some developers do not like to have their software criticized by others, and (3) some developers regard their software as flawless and therefore see no point in taking the time to having it peer reviewed.

The requirement that unit tests should provide 100-percent condition-decision coverage stems from an interpretation of the FDA standard. However, the current unit test coverage is much lower and the performed unit tests find too few errors in the software. These two issues seem to be interconnected. When testing is insufficient the full benefit does not show. Developers are thus de-motivated and see little point in unit testing. The project manager and the software architect are the problem owners who are responsible for this activity. It is generally acknowledged that the unit tests do not find enough errors and that everything is not tested, however, the problem has so far remained unresolved.

Integrate increment (4) and Test integrated increments (5). The software team has not yet defined an integration process or a process for testing the integrated software. The process descriptions are being formulated by one of the software testers. Currently, integration takes place concurrently with activities 2 and 3, but works independently hereof.

Understand FDA (6) and Produce work products (7). The development manual of the software team was developed on the basis of an interpretation of the FDA standards. The software requirement specification, the human-computer interaction navigation flow and the software architecture are the primary work products for the software team. These are updated continuously; but never synchronised with the iterations. The software architect is the primary responsible for maintaining these documents. The software architecture has troughout the project undergone several radical changes and was not in place until recently, which is several years into the project. This entailed many changes of the software and has made it difficult to implement an increment at each iteration into a working prototype.

Set goals and monitor performance (8) and Take action (9). Activity 8 is divided into two parts. The first part is setting the goals for and then monitoring the iterations. At the beginning of each iteration the goals are set in the iteration plan. The process is primarily monitored by visual checking of the tasks on the board and by the burn-down chart. The second part is setting the goals for and then monitoring the software project as a whole.

The overall goal of the project process is not clear to the developers in the software team. Contradicting goals are communicated from the higher levels of the project hierarchy and create confusion in the software team. The software team has not defined measures of performance for monitoring the software project. This issue is properly connected to the contradicting goals. Action is taken when the goal for an iteration is not reached, then the remaining tasks are moved to the next iteration. Actions are taken to improve the software project on the basis of a feeling that something needs to be improved, but this is not based on monitoring systematically.

The Impact of the Project Structure

The software team is only a small part of the entire project with more than 100 people being involved in different roles. The overall management structure is bureaucratic and the software team is the only part of the organization using an agile methodology. This influences the software development process. The hardware group for example, which the software team works closely with, works primarily sequentially as the hardware takes time to develop and each prototype is very expensive. The hardware has therefore not been available for the software team until late in the project. The early software has consequently only been tested on a simulator at the end of the iterations.

The coordination of the device project takes place on two levels: (1) at the manager level and (2) at the developer level. At the manager level the coordination takes place at a monthly device project meeting. The software project manager represents the software team in this meeting. This coordination suffers from too many stakeholders thus leading to very difficult decision-making. The software team depends on the decisions to be taken and they are the most interested in effective coordination because they are more flexible that other groups. At the developer level the coordination takes place when the developers talk to each other. The software team primarily needs to coordinate with the hardware group and the human-computer interaction group. As these are placed in the same building this coordination is relatively easy. However, they have limited understanding of how these two other groups work and what they are working on, and this limits the coordination.

The software requirements have changed several times during the project. This has mainly been due to the discovery that two requirements contradicted each other or due to the device project management changed its mind about a feature. The requirements are easy to change in the SRS and in the sprint backlog. However, it is very difficult to change a requirement in the higher-ranking documents because many stakeholders from different groups have to agree on this change.

The fact that the system test is not done by the software team, but by the system engineering group, limits the flexibility of handling the errors found at this test. When an error is found at the system test a new entity is created in the error reporting system. This reporting system handles the process of correcting the error. However, this process is very long and time-consuming as many developers have to validate and approve the changes before the error can be corrected and deleted from the task list. It is opposite with errors found at the unit tests performed by the software team as they are handled in a less formal way and the documentation load is kept at a minimum. To ensure agility in the process of handling errors it would therefore be advantageous to find as many errors as possible before releasing the software to the system engineering group.

DISCUSSION

The software team has successfully implemented parts of the agile methodology in their mixed software development approach. They are facing several problems both due to internal issues in the software team and external issues caused by the project structure. The internal and external issues are highly intertwined and their possible solution similarly interdependent. In this section the results from the analysis are discussed and their possible alleviation are related to the literature.

As the software team has no direct customer contact and receives requirements from the device project management this part of the approach becomes very document-driven. This is very much in line with traditional requirements engineering (e.g., Kotonya & Sommerville 1998; Sawyer et al. 1999). Theories of agile methodology, on the other hand, recommend close customer contact throughout the development of the product to ensure that the end-product meets the needs and expectations of the customer (Schwaber & Beedle 2001; Cockburn 2002; Beck & Andres 2004). As the requirements currently only are processed by the software architect on the basis of written project documents before entering the software backlog there is a high risk that misunderstandings occur and the product will not fit the needs of the end-user. However, this product is developed for a market and is not ordered by a known customer. This makes it difficult or impossible for the software team to create such contact on its own. But to improve the agility of the process of dealing with the requirements one or several persons from the marketing group could be assigned the role of a user (e.g., Grudin 1994). As the marketing group have in part developed the requirements in the project documents on the basis of a user survey, they are the people inside the device project having the best overview of the user needs. They could therefore serve as a middleman between the users and the software team and present the results of the user survey and ideas of the marketing group to the software team for example in the form of user stories. This suggestion is not a simple solution to this issue as the marketing group currently do not have allocated resources for it and the device project management have yet to recognize the need for more agility in this overall device project before assigning more resources.

The software team is expected to perform time-boxed iterations with a fully implemented and tested increment as output. However, the increments are often not fully tested before they are moved to the system engineering group. Testing is crucial in agile development. Most notable in XP, testing is a core practice and development is generally test-driven (Beck & Andres 2004). It is not only suggested that test cases are written before the software; it is also most importantly suggested that there is testing in of all parts of an increment and that completion is determined by the increment passing all tests. The software team does not perform unit tests to an adequate degree and their unit tests do not find a sufficient amount of errors. The connection seems to be that when the amount of unit tests is not sufficient the software team does not experience the full benefit of the tests and then the unit tests seem for them to be not worth the effort. As the tests do not find enough errors the developers are de-motivated to test further. The developers recognize that the unit tests do not work properly and that the software is not sufficiently tested. However, many of the developers do not see any value in unit testing and the implementation of new functionality seems to be valued much higher than testing. The task board and the burn-down chart give a nice overview of the remaining tasks of the iteration. However, developers also seem to be emphasising new functionality prior to testing as this is the only parameter, which the progress of the iteration is measured according to. This results in a too low quality of the software, in low stability and in several errors, when increments are delivered to the system engineering group. This is a serious issue and perhaps the main problem facing the software team. Errors detected at the system test have to be handled much more formal than the errors detected at the unit tests and integration tests, which decreases the agility. It also means that it is very difficult to determine whether an increment is complete and the iteration has been ended properly. Resolving this issue is not easy, as it will require a significant change of the developers' attitude to increase and improve the unit tests.

Initially and until recently the software team followed the recommendations of Scrum and ran iterations of the length of 30 days (Schwaber & Beedle 2001). As the increments of the software team often did not get fully tested it was decided by the project manager and the software architect to expand the iterations with one week, giving the developers 4 weeks to develop the tasks and one additional week to stabilize the software. At the time of this case study the software team had not yet finished a 5-week iteration and they were therefore not able to reflect on whether this is a solution to the problem. Unstable software did force the software team from time to time to break off from the iterations and focus on stabilization instead. This contradicts the intentions with Scrum. The problems of getting the increment fully tested taken together with the developers expressing frustration of having to implement a full increment during 4 weeks was an indication that the iterations would have to be longer. Further, the software is complex due to its safety criticality and because it is embedded in unique hardware; hence the amount of written documentation is higher than in regular agile development projects.

It was frequently mentioned during the interviews that the estimates of the iterations often do not fit, as the tasks took longer than expected or other important tasks were being pushed into the time-boxes. When the estimates did not fit some tasks were postponed to the next iteration. It seems that the estimates did not fit due to several reasons: (1) the developers do not seem to fight to reach the goal of the iteration as they do not see the importance anymore, (2) at the planning session the developers are positive and push everything they can into the iteration leaving no time for interruptions, and (3) there are many interruptions from outside the software team. In the agile methodologies interruptions during iterations are not allowed, however, in this case several of the interruptions have to be dealt with as they are serious matters affecting the whole device. The interruptions caused by severe errors in the existing software would be reduced if the quality of the increments were increased. However, the interruptions from other project groups working sequentially undergo changes in requirements as all other projects and these are more difficult to align with. This affects the software team in a significant way. This resembles the difficulty experienced by agile software teams when trying to align with organisational change processes, which are sequential too (Bygstad & Nielsen 2003; Bygstad et al. 2009). This can only be resolved if the management of the larger project understand the potential of agile software development.

The peer reviews are considered useful by the developers and several of them wish peer reviews were done more frequently. The reasons that peer reviews are not done more often anyway seem to be: (1) some of the developers have the opinion that their own software is of high quality and that they will not gain anything from a peer review, (2) some developers feel a great ownership of their software and fear getting it critiqued, and (3) new functionality has high value and the time for peer reviews is not considered when the task is estimated. As with the issue of testing the increment, this issue also seems to need a change of attitude of the developers. They need to focus more on the quality of the software instead of the number of finished tasks. Peer review is a document-oriented technique where peers read the text and assess its quality (Freedman & Weinberg 1982; Weinberg 1992). Whether this can co-exist with the emphasis on agile methodologies cannot be answered by this case study. If the software team would peer review many their software and the documentation hereof they would immediately adhere better to the FDA requirements, but the linkage between the documentation and the software is so far not really traceable.

Taken together, there are aspects of the development approach being practiced by the software team that is clearly agile (though there are parts they are not practicing well). There are also aspects that are clearly practiced in a document-driven way (which they have to practice well to remain FDA compliant). In addition, the linkages between the agile parts and the document-driven parts are not well performed. This includes: (1) little or no feedback from iteration planning (Activity 2) to producing the FDA-compliant documentation (Activity 7), (2) integration and integration testing are not performed (Activities 4 and 5), and severe interference from the outside sequential activities into the iterations. Hence, the mixed approach is a truly mixed approach, but it is not an integrated approach. In particular, it is not an integrated approach where the agile and the document-driven parts are aligned.

CONCLUSION

In this paper we have studied the software development process of a software team in a pharmaceutical company developing a highly safety critical embedded software system for drug dispensing. The software and the device in which it is embedded are therefore to be compliant with the FDA standards. To answer our research question whether agile software development is compatible with the document-driven approach, we have developed a model encompassing both the agile and the document-driven aspects and we have used that to evaluate the development practice of the software team.

The mixed software development process was designed to combine these two approaches. The software team is at the same time agile and compliant with large parts of Scrum as they run time-boxed iterations with an increment as the output. The tasks are controlled by the use of product and sprint backlogs, by the use of a task board and by the use of a burn down chart. The iterations are coordinated through daily stand-up meetings. The software team is however also document-driven as the requirements are fixed early in the project and there is no customer contact or validation of the interpretation of the requirement before implementation. The iterations are often interrupted and sometimes broken off. The load of work products is furthermore high as FDA and the project requires documentation of the development process.

This study proves that it is in one sense possible for the agile and document-driven approach to co-exist; however, they are not easily integrated. We find that the agile process suffers from its adaptation to the FDA standard, the bureaucratic management of the project and the sequential work processes of the other project groups, which cause interruptions within the iterations, changes to the hardware and user interface that the software has to cooperate with. This implies that the software team experience trouble implementing a fully tested increment during the iterations and it leaves the software unstable. To improve the stability of the software iterations are sometimes cut off for a period. At the same time many of the genuine qualities of agile development have been maintained. First, due to the use of iterations the software team was able to create running software early in the project for the management and marketing to evaluate the device early. Secondly, the use of self organizing teams, where the developers choose their own tasks results in a high degree of ownership and commitment for each task. Third, the task board and the burn-down chart are useful for visualizing the tasks and the process for the developers. Fourth, the stand-up meetings give everyone an overview of the progress of the iteration and make sure that no one is stuck in a problem for too long.

It is left as an open question how a more integrated approach could be practiced. This study rather shows the limitations of a mixed approach. Further research would be needed to investigate what characterizes an integrated approach, which parts can be integrated and to what extent such an approach would still be compliant with the FDA requirements for software development.

REFERENCES

Aaen, I., "Software Process Improvement: Blueprints versus Recipes," *IEEE Software*, (20:5), 2003, pp. 86-93.

Abdeen, M. M., Kahl, W., & Maibaum, T., "FDA: Between Process & Product Evaluation," in *Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability*, IEEE Computer Society Press, Los Alamitos, 2007.

Beck, K. & Boehm, B., "Agility through discipline: A debate," *Computer*, (36:6), 2003, pp. 44–46.

Beck, K. & C. Andres, *Extreme programming explained: embrace change,* Addison-Wesley Professional, 2004.

Bygstad, B. & Nielsen, P. A., "The Meeting of Processes," in *Proceedings of the 26th IRIS*, Helsinki, 2003.

Bygstad, B., Nielsen, P. A., & Munkvold, B. E., "Four integration patterns: a socio-technical approach to integration in IS development projects," *Information Systems Journal*, (19), 2009.

Checkland, P. & J. Scholes, *Soft Systems Methodology in Action,* Wiley, Chichester, 1990.

Chrissis, M. B., M. Konrad, & S. Shrum, *CMMI: Guidelines for Process Integration and Product Improvement,* Addison Wesley, 2003.

Cockburn, A., *Agile software development: software through people,* Addison-Wesley Professional, 2002.

FDA., (2002). Guidance for Industry, FDA Reviewers and Compliance on Off-The-Shelf Software Use in Medical Devices. Retrieved July 10, 2009, from http://www.fda.gov/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/ucm073778.ht m

Freedman, D. P. & G. M. Weinberg, *Handbook of Walkthroughs, Inspections, and Technical Reviews,* Little, Brown and Company, Boston, 1982.

Fritzsche, M. & Keil, P., "Agile Methods and CMMI: Compatibility or Conflict?," *e-Informatica Software Engineering Journal*, (1:1), 2007.

Grudin, J., "Groupware and social dynamics: eight challenges for developers," *Communications of the ACM*, (37:1), 1994, pp. 92–105.

Jakobsen, C. R. & Johnson, K. A., "Mature Agile with a Twist of CMMI," in *Agile'08*, 2008.

Jones, M., Gomez, E., Mantineo, A., & Mortensen, U. K., "Introducing ECSS Software-Engineering Standards within ESA," *ESA bulletin)*, 2002, pp. 132–139.

Knight, J. C., "Safety Critical Systems: Challenges and Directions," in *ICSE'02*, 2002.

Kotonya, G. & I. Sommerville, *Requirements Engineering: Processes and Techniques,* Wiley, Chichester, 1998.

Lee, I., Pappas, G. J., Cleaveland, R., Hatcliff, J., Krogh, B. H., Lee, P., Rubin, H., & Sha, L., "High-confidence medical device software and systems," *Computer*, (39:4), 2006, pp. 33–38.

McMichael, B. & Lombardi, M., "ISO 9001 and Agile Development," in *Agile 2007*, IEEE, 2007.

Mutafelija, B. & H. Stromberg, *Systematic Process Improvement Using ISO 9001:2000 and CMMI,* Artech House, 2003.

Opelt, K. & Beeson, T., "Agile Teams Require Agile QA: How to make it work - an experience report," in *Agile'08*, 2008.

Rakitin, R., "Coping with defective software in medical devices," *IEEE Computer*, (39:4), 2006, pp. 40-45.

Rottier, P. A. & Rodrigues, V., "Agile Development in a Medical Device Company," in *Agile'08*, 2008.

Sawyer, P., Sommerville, I., & Viller, S., "Capturing the benefits of requirements engineering," *IEEE Software*, (16:2), 1999, pp. 78-85.

Schrenker, R. A., "Software engineering for future healthcare and clinical systems," *IEEE Computer*, (39:4), 2006, pp. 26-32.

Schwaber, K. & M. Beedle, *Agile software development with Scrum,* Prentice Hall PTR, Upper Saddle River, NJ, 2001.

Sutherland, J., Jakobsen, C. R., & Johnson, K., "Scrum and CMMI Level 5: The Magic Potion for Code Warriors," in *AGILE 2007*, 2007.

Theunissen, W. H. M., Kourie, D. G., & Watson, B. W., "Standards and Agile Software Development," in *SAICSIT 2003*, 2003.

Turner, R. & Jain, A., "Agile Meets CMMI: Culture Clash or Common Cause?," in *XP/Agile Universe 2002*, D. Wells & L. Williams, editors, Springer, Berlin, 2002.

Walsham, G., "Interpretive Case Studies in IS research: Nature and method," *European Journal of Information Systems*, (4), 1995, pp. 74-81.

Weinberg, G. M., *Quality Software Management,* Dorset House Publishing, New York, 1992.

COPYRIGHT

INTRODUCING AGILE PRACTICES IN A DOCUMENTATION-DRIVEN SOFTWARE DEVELOPMENT PROCESS: A CASE STUDY

Lise Tordrup Heeager
Aalborg University, Denmark
liseh@cs.aau.dk

ABSTRACT

The agile and the documentation-driven software methods advocate two different ways of developing software and therefore seem contradicting; but research does show compatibility between these. Practitioners have an increasing desire to adopt agile software development methods while still being compliant with a quality assurance standard. However due to little empirical research documenting this compatibility, it is not well understood how to implement a software development process meshing the agile and the documentation-driven methods. This paper presents a case study of a software development process in a pharmaceutical company. For market reasons the company has to comply with the US Food and Drug Administration standards for software development. At the same time significant parts of the agile method Scrum have been implemented. The purpose of this paper was to understand the possibility of implementing a software development process, which meshes the agile and the documentation-driven methods and the challenges of doing this. This case study supports the previous literature and proves that it is possible to implement a software development process, which meshes the agile and the documentation-driven methods. It shows that it is possible to mesh the project characteristics: planning and control, communication and knowledge sharing and environment. It was however also proven difficult to mesh the project characteristics: developers, requirements, testing and customer-relations.

Keywords: *Agile Software Development, Documentation-Driven Software Development, Quality Standards, Scrum, FDA*

INTRODUCTION

In information systems development, an increasing amount of practitioners are interested in adopting methods (Avison, Fitzgerald 2003), and the matter of choosing the appropriate software method is occupying many IT practitioners (Fitzgerald et al. 2002). Within the development of information systems, the agile and the documentation-driven methods advocate two different approaches to software development (Dahlberg et al. 2006). The agile software methods (e.g. XP and Scrum) promise to deliver software without excessive cost (Beck, Andres 2004; Schwaber, Beedle 2001); they are based on the assumptions that programs and programming is primary, not documentation. Agile development seeks to increase software quality by adhering to a very flexible process (Cockburn, Highsmith 2002). The documentation-driven software methods are most often associated with the Waterfall model, but is also represented by quality standards, for example the CMMI (Chrissis et al. 2003) and the ISO standards (Hoyle 2006). They rely on the assumption that to achieve high quality, software development processes must be defined and documented, requirements are established and documented before designing, designs are established and documented before programming, testing is thorough and documented, and there are procedures through which documentation can be changed.

Neither of the agile nor the documentation-driven methods provide the ultimate approach, both has strengths and weaknesses (Boehm, Turner 2003a). The documentation-driven methods work well for stable projects, but they lack the ability of handling change. The agile methods embrace change (Beck, Andres 2004), but offhand they lack the ability of quality assurance. Software organizations have therefore become increasingly interested in meshing the agile and the documentation-driven methods (Pikkarainen, Mäntyniemi 2006), in order to create a software process, which takes advantage of the strengths and compensate for the weaknesses of the two (Galal-Edeen et al. 2007). Even though the agile and documentation-driven methods seem contradicting, several researchers agree that they are compatible (e.g. Baker 2005; Heeager, Nielsen 2009; Manhart, Schneider 2004; Nawrocki et al. 2006). Due to the differences between the agile and the documentation-driven methods meshing the two is not straightforward and it is not well understood how to do this (Reifer 2003; Vinekar et al. 2006). At this point only little empirical research documents the compatibility of the agile and documentation-driven methods and how to implement a meshed process (Kähkönen, Abrahamsson 2004; Pikkarainen 2009) - especially not on how to mesh the agile and the documentation-driven methods when developing safety-critical software (Vogel 2006).

To fill this knowledge gap, this paper provides a detailed understanding of how a software group has implemented and improved a software process meshing the agile and the documentation-driven methods. The purpose was to understand the possibility of implementing a software development process, which meshes the agile and the documentation-driven methods and the challenges of doing this. This paper is based on a longitudinal, interpretive case study of a software development process used by a software group in a large, pharmaceutical company. The software group is developing a clinical, highly safety-critical product with embedded software, which by law has to comply with process standards such as the US Food and Drug Administration (FDA) standard for software development. But, at the same time they have introduced several practices of the agile method, Scrum. The analysis gives an overview of the changes and improvement of the software process identified at the two different times of data collection.

In the following section the theoretical background is presented and the analysis framework developed. In the third section the research method is outlined. This is followed by a description of the case. Then the analysis and evaluation of the software development process is presented. Finally the results are discussed and concluded on.

THEORETICAL BACKGROUND

This section presents the theoretical background used to create the analysis framework.

The Agile and the Documentation-Driven Methods

The agile methods were developed based on the shortcoming and failures of documentation-driven software methods. The agile manifest (developed in 2011) expresses some of the differences between the agile and the documentation-driven methods: The agile methods value all the items, but puts more value on the items on the left; the items on the right are associated with documentation-driven software methods: 1) individuals and interactions over processes and tools, 2) working software over extensive documentation 3) customer collaboration over contract negotiation and 4) responding to change over following a plan (Beck et al. 2001). Agile methods advocate an iterative, test-driven software development process with frequent customer feedback. Short iterations, multi-disciplinary teams, knowledge sharing and continuous integration allow (better) control over the project and increase visibility (Mahanti 2004). Agile methods also advocate an informative workspace, in which the development is supported by information radiators (the placement of relevant information where passers-by can see them) (Cockburn 2006). Refactoring (redesigning and rewriting software) is also advocated by the agile methods, as it serves to correct for poorly written and redundant code (Vogel 2006). eXtreme Programming (XP) (Beck, Andres 2004) and Scrum (Schwaber, Beedle 2001) are the two most well-known agile methods. While XP offers a range of practices and principles to apply in order to create flexibility and adaptability, Scrum provides guidance for efficient management of projects (Jakobsen, Johnson 2008). Scrum operates with three roles and several practices (see table 1).

Role/practice	Description
Scrum master	The Scrum Master is responsible for the scrum process.
Product owner	The product owner is the customer.
Scrum team	Self-organizing project teams.
Sprints	The team works for a fixed period of time called a sprint (30 days is recommended).
Daily scrum meeting	Each scrum team meets daily for a 15-minutes status meeting; these meetings are often done standing up.
Sprint planning meeting/day	The next sprint goal and functionality are determined at the sprint planning meeting. The scrum team then devises the individual tasks to be performed.
Sprint review meeting	The scrum team presents the product increment built during the sprint.
Product backlog	A prioritized list of all product requirements.
Release backlog	The subset of the product backlog selected for a release.
Sprint backlog	The tasks devised for a sprint.
Scrum boards	Boards displaying the tasks chosen for the current iteration and their status.

Table 1. Practices and Roles of Scrum (Schwaber, Beedle 2001)

The documentation-driven methods strive at creating predictability, repeatability and optimization; the methods aim at achieving this by focusing on long-term plans controlled by long-term milestones (Boehm 2002). They rely on a codification strategy in which documentation of the process and product is produced (Nerur et al. 2005). The project is controlled through a command-and-control style of management, which includes a clear separation of roles (Moe et al. 2010). The customer is included in the beginning of the project, where the requirements are specified and the contract is negotiated (Nerur et al. 2005). Documentation-driven methods are often associated with the Waterfall model consisting of sequential development stages. Besides from the Waterfall model, documentation-driven methods are also associated with different quality standards such as the CMMI (Chrissis et al. 2003) and the ISO9000 family of standards (Hoyle 2006). For safety-critical software development even stricter quality standards must be applied; one example is the FDA standard. FDA approval relies on a process-driven approach. The manufacturers of the software obtain approval by showing that they have applied quality assurance techniques to certain levels of coverage (Lee et al. 2006). The development process therefore include: 1) identification of the system hazards and a definition of safety requirements, 2) a determination of how the various components in the system can contribute to these hazards, 3) defining derived safety requirements for the components (repeat recursively over the system hierarchy as needed), and then 4) the development of the components to meet these safety requirements (Heimdahl 2007).

Differences between the Agile and the Documentation-Driven Methods

The literature shows that the agile and the documentation-driven methods have some differences in their focus, in their customer-relations, in the amount of documentation, in their development strategy and in the way they handle requirements. People-issues are at the heart of the agile methods (Galal-Edeen et al. 2007) as they focus on the social aspects of software development (McAvoy, Butler 2009). It is a significant departure from the focus on processes suggested by the documentation-driven methods (Vinekar et al. 2006). Both the documentation-driven and the agile methods highly value the satisfaction of the customer. But, in documentation-driven methods the customer is mainly involved during the early phase of the project, while agile methods involve the customer throughout the whole development process (Paetsch et al. 2003). The handling of requirements is also a difference between the agile and the documentation-driven methods. While the agile methods bases the requirements on user stories created and constantly updated by the customer (Beck, Andres 2004), the documentation-driven methods advocate that the requirements are established and documented before designing and so forth (Pressman 2000). The amount of documentation is one of the most significant differences between agile and documentation-driven methods; agile methods do not support the degree of documentation demanded by documentation-driven methods (Boehm, Turner 2005; Paetsch et al. 2003). Agile methods advocate an iterative software process model (Larman 2004), which opposed to the documentation-driven methods (which have separate coding and testing phases) include the testing and debugging of the written code in each iteration (Cohn, Ford 2003).

Compatibility of the Agile and the Documentation-Driven Methods

Previous research agrees that the agile and the documentation-driven methods despite the differences to some degree are compatible, but disagree about the level of compatibility. Based on a literature study XX (forthcoming) concludes that it is possible to implement several agile principles and practices in a software development process and at the same time comply with a quality assurance standard, but no evidence shows that it is possible to implement a full agile software development process in a documentation-driven setting.

Some researchers argue that the documentation-driven methods support the agile methods by providing a disciplined framework; CMMI for example presents a framework for process improvement with guidelines of what to do to improve (Elshafey, Galal-Edeen 2008). The documentation-driven methods can furthermore help identify the shortcomings of the agile methods by checking the coverage of each process area; the potential for improvement can thereby be identified (Fritzsche, Keil 2007). The ISO standard has been used to provide structure to a software process and help ensure the agile processes were followed (McMichael, Lombardi 2007). On the other hand agile practices ensure that processes are implemented efficiently while embracing change. The agile method contributes with adaptability, while the documentation-driven method contributes with discipline (Jakobsen, Johnson 2008; Sutherland et al. 2007).

Both the agile (Theunissen et al. 2003) and the documentation-driven (Baker 2005) methods are highly flexible and adaptable. Agile methods can be adapted to ensure compatibility with standards, such as the ISO. Wright (2003), for example, concludes that companies using XP can meet the requirements of standards such as ISO9001; the key is to base the quality management on the natural outputs of the development process and not rely on work products produced for the sake of the quality assurance. Research also compares the agile methods and CMMI. There is a consensus that CMMI level 2 is largely compatible with the agile methods, whereas CMMI level 3 and agile methods are partially compatible. Comparing the process areas of CMMI with the agile methods, Alegria and Bastaricca (2006) find that the process areas of CMMI level 2 is 72% fulfilled, whereas the CMMI level 3 is only 60% fulfilled. XP and CMMI level 2 are highly compatible, because both XP and CMMI focus on basic project management; but an extension of XP is however needed to fulfil all requirements (Bos, Vriens 2004). XP fulfils many of the CMMI level 3 requirements as well; XP for example includes mechanisms for verification and criteria for completion (Laurila 2004). Comparing the process areas of CMMI level 4 and 5 and agile methods, they seem to be less compatible; XP needs management and infrastructure (Paulk 2001). A few case studies however show that it is also possible to comply with CMMI level 4 or 5 (Anderson 2005; Sutherland et al. 2007).

The Home Grounds of the Agile and the Documentation-Driven Methods

The home ground concept was introduced by Boehm (2002); a home ground is project characteristics within which a method performs very well. As the home grounds compare the agile and the documentation-driven software methods on general project characteristics, they are useful for analysing a specific software process in order to determine if and how it meshes the agile and the documentation-driven principles and practices. This paper will therefore adopt the home grounds as the analysis frameworks (table 2 and table 3). The descriptions of each characteristic (referred to as a category) and its sub category is based on the literature on the agile and the documentation-driven methods. Both the agile and the documentation-driven home grounds consist of 12 characteristics divided into 4 categories: personnel, management, technical and application (Boehm, Turner 2003b).

Construct	Category	Subcategory	Description
Agile	Personnel characteristics	Agile developers	Critical people-factors for agile methods include amicability, talent, skill, and communication.
		Agile customers	Agile methods strongly depend on dedicated, collocated customers.
		Agile culture	In an agile culture, the people feel comfortable and empowered when there is a high degree of freedom.
	Management characteristics	Agile customer relations	Agile developers use working software and customer participation to instil trust in their track record, the systems they have developed, and the expertise of their people.
		Agile planning and control	Agile methods see planning as a means to an end rather than a means of recording in text.
		Agile communications	Agile methods rely heavily on tacit, interpersonal knowledge.
	Technical characteristics	Agile requirements	Most agile methods express requirements in terms of adjustable, informal stories.
		Agile development	Agile methods advocate simple design, one that emerges as functionality is implemented.
		Agile testing	Agile methods use executable test cases, which define requirements, the development is test driven.
	Application characteristics	Agile size	Agile methods seem to work best with small to medium teams of people working on relatively small applications.
		Agile primary objective	The primary goals of agile methods are rapid value and responsiveness to change.
		Agile environment	Agile methods are most applicable to turbulent, high change environments.

Table 2: Analysis framework (agile characteristics), adapted from Boehm (2002)

Construct	Category	Subcategory	Description
Documentation-driven	Personnel Characteristics	Documentation-driven developers	Documentation-driven methods of course do better with great people, but generally focus on planning the project and defining the architecture of the software so that less-capable people can contribute with low risk.
		Documentation-driven customers	Documentation-driven methods rely on contractual plans and specifications written and agreed on in the beginning.
		Documentation-driven culture	In a documentation-driven culture, the people feel comfortable and empowered when there are policies and procedures that define their role in the enterprise; each person's tasks are well-defined.
	Management Characteristics	Documentation-driven customer relations	Documentation-driven methods generally depend on a contract between the developers and customers as the basis for customer relations.
		Documentation-driven planning and control	Documentation-driven methods rely on documented plans and a project manager in charge of the control.
		Documentation-driven communications	Documentation-driven methods rely heavily on explicit documented knowledge.
	Technical Characteristics	Documentation-driven requirements	Documentation-driven methods use formally base lined, complete, consistent, traceable, and testable specifications.
		Documentation-driven development	Documentation-driven methods rely on extensive design.
		Documentation-driven testing	Documentation-driven methods use documented test plans and procedures. They use late-testing.
	Application Characteristics	Documentation-driven size	Documentation-driven projects scale best to large projects (Large teams and products).
		Documentation-driven primary objective	The primary goals of documentation-driven methods are predictability, stability, and high assurance.
		Documentation-driven environment	Documentation-driven methods work best when the requirements are largely determinable in advance and remain stable.

Table 3: Analysis framework (documentation-driven characteristics), adapted from Boehm (2002)

RESEARCH APPROACH

This is a longitudinal, interpretive single case study (Walsham 1995) of how a safety-critical product with embedded software is developed at a pharmaceutical company. The purpose of the study was to evaluate the software development process used by the software group in one of their projects. To be able to identify changes in and improvements of the software process, the empirical data was collected and validated iteratively through two phases (Table 3). Each phase consists of 3 sub phases in order to introduce an iterative approach of data collection, analysis and validation. This process of data collection stretched over a period of approximately 2 years.

Phase	Sub phase	Duration	Data collection	Data documentation
1	a	November 2008 – June 2009	7 qualitative interviews based on an interview guide	Interviews audio recorded and transcribed
	b		8 (4+4) qualitative interviews based on 2 interview guides	Interviews audio recorded and transcribed
	c		Seminar presentation and validation with 11 members of the software group	Meeting audio recorded
2	a	January 2010 – August 2010	7 qualitative interviews based on an interview guide	Interviews audio recorded and transcribed
	b		5 qualitative interviews based on an interview guide	Interviews audio recorded and transcribed
	c		Observations of a planning meeting and a stand-up meeting	Meetings audio recorded and transcribed

Table 3: Phases in the interpretive research approach

The primary data collection technique was interviews as it in qualitative research is an important data gathering tool (Myers, Newman 2007). It is through the interviews, the researcher has the best opportunity to access the interviewee's interpretations of the actions and events, which have or are taking place (Walsham 1995). The interviewees were chosen in order to include managers, developers and testers. This way each sub team of the software group was represented. A triangulation of the data was made as the interviews were supplemented by observations and document studies.

Data Collection: Phase 1

The first phase included two sub phases of interviews (1a + 1b) and a sub phase with a seminar presentation and validation of the results so far (1c). All interviews were organized as diagnostic interviews (Iversen et al. 1998), based on a semi-structured interview guide. In the first sub phase (1a) 7 qualitative interviews were conducted with: the project manager, the software architect, a software tester and four developers from both sub teams. The interview guide included the subjects: FDA requirements, agile software development, the requirement specifications, handling of errors and general strengths and weaknesses. The second sub phase (1b) included eight interviewees: the project manager, the software architect, a tester and five developers from both sub teams. Four of these interviewees participated in the first phase, whereas the last four participated for the first time. Two interview guides were used. Interviewees participating for the second time were asked questions based on the activities identified in sub phase 1a and concluded by validating some of the statements from the first interviews. New interviewees were asked questions which included the subjects of FDA requirements and agile development before discussing the activities and evaluating the statements for the validation of the first phase. The findings were written into a report and presented to the software group in a seminar including a presentation and a discussion of these. The interviewees from the previous two sub phases participated.

Data Collection: Phase 2

The second phase also included two sub phases (2a + 2b) of interviews and added a sub phase (2c) of observations in order to verify the procedure and outcomes of the planning meetings and stand-up meetings. The data collection was based on a framework of the home grounds of the agile and the documentation-driven methods (Boehm 2002). The interviews were semi-structured by an interview guide based on the characteristics of the home grounds. The purpose was to follow up on the evaluation from phase 1 and to identify changes in and improvement of the software process. The first sub iteration (2a) included seven interviews with: 3 software developers, a tester, the new software architect, a coordinator between the software group and other groups of the project and a consultant specializing in Scrum, who had been affiliated to the software group for 5 months. The output of the interviews was an identification and description of activities within the characteristics of the home grounds and a revised evaluation. Sub iteration 2b focused on gaining a detailed description of the activities identified in sub iteration 2a. It included 5 interviewees: the former project manager, who was reassigned as responsible for the process, the former architect, now functioning as a software developer, the new scrum master, a tester and a software developer. The purpose of sub iteration 2c was to determine the procedure and the outcome of the planning meetings and stand up meetings. It included 2 observations, 1 observation of a planning meeting and 1 observation of a stand up meeting.

The Data Analysis

In total twenty eight interviews, observations of meetings and document studies were conducted. All interviews were audio recorded and transcribed. As coding of the data provides a structured analysis process, the interviews were collected into one hermeneutic unit and coded using ATLAS.ti V6 (Muhr 1991). The coding was based on the framework of the home grounds of the agile and the documentation-driven methods, including the characteristics: agile or documentation-driven personnel characteristics, agile or documentation-driven management characteristics, agile or documentation-driven technical characteristics and agile or documentation-driven application characteristics. The codes consisted of the subcategories of the frameworks, making it easy to evaluate the software process on each project characteristic. The full analysis were written up, presented in a report and validated by the software group.

CASE DESCRIPTION

This is a study of a software group of a pharmaceutical company which developed an embedded, clinical device for dispensing drugs. Due to the high level of safety-criticality the overall device project had to be compliant with a large number of quality standards (ex. FDA), some applying directly to the software development process. The project studied is the first and by far the largest and most complicated software project in the organization.

The study focuses on the software development process. In the beginning, the software development suffered from high complexity and uncertainty of the tasks, shifting requirements and the strict rules of the FDA standard. To increase adaptability the software group decided to implement practices of the agile software development method, Scrum. This is an analysis of how they have gone from a very documentation-driven software process to a software development process, which meshes a documentation-driven FDA-compliant method and the agile software method, Scrum.

During the period of the case study the software development group expanded from 17 to approximately 30 developers, testers and managers. The software group now consists of the software project manager, a software architect, the process manager, a coordinator responsible for the coordination with the other parts of the project, the developers and the testers.

The software development process is divided into iterations which has varied in length but now has settled on 2 weeks; each iteration is initiated with a planning session, which includes a retrospective of the previous iteration and the planning of the future iteration. The developers are divided into two sub teams, which are coordinated by the scrum master. The testers are organized in their own sub team controlled by a test manager. The two sub teams of developers work on the same product backlog. The tasks of an iteration are displayed on two scrum boards on the back wall in the office. The developers primarily coordinate within the sub teams through the daily stand-up meetings. The developers are responsible for unit testing and reviewing the code during each iteration; the test team conducts the integration tests before the code is handed to the system engineering group which runs the system tests of the increments, released every second month.

The process of the test team is very different from the process of the developers. They use a highly documentation-driven development process, which is managed by the test manager. Each member has an area of which they are responsible, for example: one is responsible for the design of the test cases while another is responsible for implementing the code, and a third is responsible for automating the tests.

The product development also involves pharmaceutical and clinical researchers responsible for the drugs, process engineers responsible for production facilities, mechanical engineers responsible for the products mechanical functions dispensing the drugs, embedded hardware engineers responsible for the computer chips and communication controlling the mechanics, and software developers responsible for the software controlling the hardware. In total over 100 managers, researchers, engineers and developers are working on the device project. The software group is working in close cooperation and is dependent of the engineers developing the hardware, the mechanics and the system engineering group. Prior to this project, the organization had very little experience with software development and all software processes were to be developed from scratch. The project has been running for several years and is soon to enter the last state of refining the product.

ANALYSIS

The software group and its software development process are analyzed according to the characteristics of the home grounds of the agile and the documentation-driven methods. The analysis focuses on the changes in and improvements of the software process from the time of the first data collection (the prior software process from 2008-2009) to the second (the recent software process from 2010). During the period of study, the software development process has gone from a primarily documentation-driven software process to a software development process, which includes several agile principles and practices. But, the software group still faces several problems both due to internal issues in the software group and external issues caused by the project structure. The findings are summarized in table 4 (the prior software process) and table 5 (the recent software process). The most relevant characteristics are described in further details below.

Characteristics	Agile	Meshed	Documentation-driven
Developers			Primarily documentation-driven.
Customers			No customers affiliated.
Culture			Primarily documentation-driven
Customer relations			No relations or contact to a customer.
Planning and control		Iterations of 4 weeks length, which often do not include testing and is sometimes disrupted to ensure stability.	
Communications and knowledge sharing	The communication of the developers is primarily face-to-face.		
Requirements			The requirements were established in the beginning of the project and written in documents.
Development			The development process is regulated by formal requirements.
Testing			Much unit test and integration testing is postponed.
Size			A large software group
Primary objective			The primary objective of the developers is being approved.
Environment	The software group has created an agile environment with for example scrum boards.		

Table 4: Summary of the prior software development process

Characteristics	Agile	Meshed	Documentation-driven
Developers		A mix of documentation-driven and agile developers.	
Customers			No customers affiliated.
Culture		The culture has both agile and documentation-driven aspects.	
Customer relations		The product owner role is filled by the software project manager.	No relations or contact to a customer.
Planning and control	The software development is highly iterative, using iterations of 2 weeks length.		
Communications and knowledge sharing	The communication of the developers is primarily face-to-face.		
Requirements		The requirements specifications have been rewritten.	
Development			The development process is regulated by formal requirements.
Testing			Much testing still has to be done.
Size	The developers are divided into sub teams.		
Primary objective			The primary objective of the developers is being approved by the authorities.
Environment	The software group has created an agile environment.		

Table 5: Summary of the recent software development process

Developers

At the beginning, the organization had very little experience in software development projects. The majority of the software developers initially assigned the project had a background in production and documentation-driven software development. Many of these developers were reluctant towards the adoption of the agile practices, and had problems adjusting to the agile mindset. Acknowledging that the competences of these software developers were ill suited for the project and for the ambition of implementing an agile software process; the software developers hired during the project period had a strong software background and experience with agile software development. Additionally several resources was spent on educating the developers, for example: a consultant facilitating the scrum process was affiliated to the software group for six months. The final consequence taken by management of the software group was to remove some of the inexperienced developers from the development process, assigning them to documentation related tasks. All of these initiatives resulted in a more experienced software group, consisting of a mix of agile and documentation-driven developers (in the recent software process).

Customer Relations

As the device was developed for a market and the end-users were not associated the project, the software group had no relations or contact to a customer, neither in the prior software process nor in the recent software process. The relation to the end-user was a market analysis done in the beginning of the project by the marketing group. This made the customer-relations documentation-driven and made it difficult to fill the role of the product owner (advocated by Scrum). One solution would have been to assign the product owner role to a person in the project with a full overview of the requirements of the device (for example a person from the marketing group). In order to do that the software group tried to convince the management that a detached product owner was needed, but without any luck. Instead, the software group had (in the recent software process) split the role and assigned the role as official product owner to the software project manager and the role as technical product owner to the software architect. Neither the software project manager nor the software architect however had an adequate overview of the product and its requirements to take the decisions expected of a product owner. It had furthermore unfortunate complications to have a product owner who was also designing and implementing the system; no one saw the product with the eyes of the customer. As the requirements (based on written project documents) only were processed by the software architect before entering the product backlog the risk that misunderstandings occurred and the product did not fit the needs of the end-user was heightened.

Planning and Control

The purpose of introducing Scrum was to introduce adaptability in the software development process to cope with the complexity, uncertainty and shifting requirements. Initially (in the prior software process) the software group followed the recommendations of Scrum and implemented iterations of 4 weeks length. The developers reluctant to the idea of an agile process were skeptical towards introducing iterations and had trouble breaking the tasks down into smaller bits, which could be implemented and tested within each iteration. Furthermore, while the software group tried to use iterative development, the project as a whole was controlled by long-term plans and milestones. The software group was influenced by these milestones as they determined the goals and activities for each period in the project and therefore highly influenced the priorities and focus. The iterations were also affected as interruptions from other project groups happened regularly, even though the scrum master tried to enforce the agile recommendation of not letting anything interrupt the developers in the middle of an iteration. Several of these interruptions had to be dealt with, as they were serious matters affecting the whole device. The amount of urgent interruptions several times forced the software group to break of the iteration concerned. Some of these interruptions were caused by severe errors in the existing software discovered at the system test run at a release and from changes demanded by other project groups working sequentially.

After having used iterative software development for a longer period and the number of agile developers had increased, several of the developers began to see the advantages of iterative development. Lead by the agile consultant, the developers decided to decrease the iteration length to 2 weeks (in the recent software process). These short iterations had several advantages. It gave the developers a steady work rhythm and a dedicated focus, as they constantly were forced to focus on the highest prioritized tasks. The short iterations also forced the developers to break the tasks down into even smaller bits, which gave an even greater transparency of the content of the tasks. This had a positive influence on the developers' ability to estimate the tasks.

Communications and Knowledge Sharing

The communication and knowledge sharing between the developers were primarily face to face, both in the prior and in the recent software process. The developers coordinated daily at the stand up meetings; at these meetings the current tasks and the progress of the iteration were discussed. In the beginning of the project the developers were very skeptical towards the stand-up meetings. Before the software group was divided into sub teams, the whole group participated in the same meetings, which due to the growing size of the group became very long and partly uninteresting. After the division of the software group and after the developers gained experience in conducting stand-up meetings, several developers however, did find great value in these meetings as they ensured that all members of the sub team kept a shared focus and gave the developers a frequent overview of the progress of the iteration. The planning meetings and the including retrospectives also functioned as forums for communication and knowledge sharing about the tasks and the iteration progress. Each sub team of developers had its own planning meetings and stand up meetings; the coordination between the teams was facilitated by the scrum master. An informative work space in which the scrum boards played a central role was also a primary communication channel. At the scrum boards the tasks of an iteration and their status were easily accessed. At a weekly status meeting the progress was communicated to the entire software group and its management. These meetings were supported by another board focusing on whether or not the development process followed the overall plan. The writing of documents played a smaller role in the knowledge sharing within the software group, as new documentation was distributed less frequently and found uninteresting by most of the developers. The development had however still a great focus on documents as these were required for the approval of the product.

Requirements

The requirements were established in the beginning of the project based on a market analysis. The requirements were written down and communicated through a hierarchy of documents. The higher the document was placed in the hierarchy, the harder it was to change the requirement.

Requirements were also developed based on a hazard analysis which revealed the safety hazards of the device. The software requirements written in the software requirements specification were by the software group transformed into a product backlog with prioritized and estimated tasks. These tasks were by the developers referred to as user stories, but they were not written by users and were not in the form as user stories proposed by agile methods.

Problems with the requirements resulted in a rewriting of the documents late in the development process (at the time of the recent software process). Several of the higher requirements specifications were filled with requirements which were not specified properly, with requirements which were too specific and with conflicting requirements. This was a problem for the software group as they experienced difficulties interpreting and implementing the requirements. The software developers did not have the authority or knowledge to sort out conflicting requirements or fill in the missing details. In the end, the process manager therefore pushed a rewriting of these documents through. This rewriting of the requirements moved the project characteristic requirements towards a more meshed process in which requirements are improved during the development, however not in an highly, iterative and agile way.

Testing

Both in the prior and in the recent software process, the software group tried to implement time-boxed iterations with a fully implemented and tested increment as output. But, a lot of the testing was postponed until later in the project, the increments were therefore not fully tested before they were moved to the system engineering group. The quality of many of the unit tests implemented was also too low. The lack of testing contributed to a low quality of the code, in terms of low stability and in several errors when increments were delivered to the system engineering group. Errors detected at the system test (performed by the system engineering group) had to be handled much more formally than the errors detected at the unit tests and integration tests. The high amount of errors found at the system test therefore decreased the agility.

In the recent software process the developers did increase the amount of unit tests done within an iteration and the quality of each unit test, as 1) the skills of the developers were heightened; 2) the developers experienced the benefits of unit testing and were motivated to increase the amount of tests (a positive circle, more tests breeds more tests) 3) the value of the testing process was increased, as the agile consultant had a focus on this issue. The focus on integration test also increased heavily, and the number of testers responsible for the integration test of the code increased intensely through the project, especially in the last period. However, as the recent test team worked on a big backlog of tests, they were not able to test each increment directly after the iteration. This created a big cliff between the developers and the testers, whose cooperation was almost nonexistent at the time of the recent software process.

Environment

The software group had already in the prior software process implemented an agile environment in which people physically were placed close together in the scrum teams and surrounded by information radiators. The environment did not undergo any radical changes from the prior to the present software process. The primary information radiators were the scrum boards (one for each sub team of developers) displaying the progress of the iteration. The scrum boards were placed on the back wall of the office, visible for all the developers. Several advantages were associated with the boards: the progress and status of the iteration was constantly visible and easy accessed and the boards supported the discussions at the stand-up meetings. By adding columns symbolizing sub activities of the process such as the peer review, the developers brought these into focus and made it visible for everybody whether or not a task had undergone every necessary activity.

DISCUSSION

This case study is an example of how a meshed software process can be implemented in a documentation-driven software development process developing safety-critical software. As this case study represents the most challenging conditions for meshing the agile and the documentation-driven methods, the findings represent a minimum of compatibility and is a prove that it not only can be done in non-critical projects.

In this section the software development process of the software group is discussed according to the literature. The section highlights the project characteristics in which the agile and the documentation-driven methods have proven to be compatible and the project characteristics in which meshing the agile and the documentation-driven methods has proposed challenges. Table 6 summarizes the project characteristics, which are described in more detail below.

Project characteristics	**Description**
Compatible project characteristics	
Planning and control	It is possible to implement a highly iterative software process in a documentation-driven setting and doing so can have several advantages.
Communication and knowledge sharing	Implementing a communication and knowledge sharing strategy which is primarily agile and personalized is possible even in a documentation-driven setting, but documentation is still needed.
Environment	Using information radiators (especially scrum boards) can be very advantageous as they provide visibility of the process.
Challenging project characteristics	
Developers	Changing the mindsets of developers can be difficult; developers trained in the documentation-driven domain may have a hard time adjusting to the agile mindset.
Requirements	It can be difficult to handle the requirements in an agile, iterative manner.
Testing	It can be difficult to implement a test-driven process.
Customer-relations	For a software group developing software for a device for the market it can be difficult to implement a customer-driven process.

Table 6. Summary of the findings.

Compatible Project Characteristics

The agile and the documentation-driven methods have proven to be compatible on the project characteristics: planning, communications and knowledge sharing and environment.

Planning and Control – Implementing an Iterative Development Strategy: It is possible to implement a highly iterative software process in a documentation-driven setting and doing so can have several advantages. One of these is an increase of adaptability. The purpose of agile software methods are to be adaptable, collaborative, delivery-driven, people oriented and customer-focused (Larman 2004). The agile software development methods such as Scrum and XP recommend iterative software development (Beck, Andres 2004; Schwaber, Beedle 2001). This case study has shown how the software group successfully has introduced an iterative software process in which the software was developed in iterations (varying in length – narrowed down from 4 to 2 weeks). Many advantages were associated with the iterations (especially the iterations of 2 weeks length); breaking the tasks down (to fit within the iterations) into a series of manageable chunks increased the transparency of the content of the tasks and eased the process of estimating. In this case the iterations were however affected by the long-term milestones guiding the overall project. The activities demanded to pass the milestones, such as documenting the requirements and the software design in the early phases influenced the workflow and the focus of the iterations and decreased the agility.

Communication and Knowledge Sharing – Implementing a Personalized Communication Strategy: A primarily agile and personalized knowledge sharing strategy can be implemented in a documentation-driven setting, but documentation is still needed. While the documentation-driven methods facilitate knowledge sharing primarily through documentation (codification), the agile methods facilitate knowledge sharing through informal communications among team members (personalization) (Boehm, Turner 2003b; Crawford et al. 2006). Previous research shows that the agile and the documentation-driven software development methods are limited in the compatibility when it comes to the process of documentation (Boehm, Turner 2005; Paetsch et al. 2003). In this case study the software group had implemented a highly agile and personalized communication and knowledge sharing strategy, as they primarily communicated at the planning meetings, daily scrum meetings and through the information radiators. To comply with the FDA standard they also produced documentation, but as a communication strategy, codification only played a small part; according to the developers writing documents was a minor part of their work and their knowledge sharing, but it did however still play a central role as they for example had to document each peer review.

Environment – Use of Information Radiators: Using information radiators (especially scrum boards) can be very advantageous as they provide visibility of the process. Information radiators are an agile strategy for displaying important information about the process. The information radiators pass along information quietly, with little effort, and without disturbing people (Cockburn 2006). The developers of this case study found it advantageous to use information radiators, especially the two scrum boards used to structure the tasks of the iteration. These boards provide a visibility of the progress of the iteration. The developers also found it advantageous to increase the number of columns representing the sub activities of solving a task. Adding such a column creates a focus on the sub activity concerned; for example adding a column representing the peer reviews increased the focus on these and contributed to an increase in the number of peer reviews done within the time limit of the iteration.

Challenging Project Characteristics

Whereas the previous project characteristics showed compatibility, the case study showed that the following four project characteristics can be difficult to mesh: developers, requirements, testing and customer-relations.

Developers - Changing the Mindsets of the Developers: Changing the mindsets of developers can be difficult; developers trained in the documentation-driven domain may have a hard time adjusting to the agile mindset. Changing from a documentation-driven to an agile process for example requires a different mindset and different skills (Mahanti 2004). In this case study, in the beginning of the project the developers were reluctant towards implementing Scrum practices. Based on an empirical study of seven organizations, Cohn and Ford (2003) also experienced that even though most developers have an appropriate level of scepticism towards the adoption of agile practices, some developers did resist the change. This case study also showed that, as the number of developers with experiences in agile software development increased more agile practices and principles were implemented. Competent people were needed to make the implementation happen, again supported by the research on agile methods, as little evidence is given, that agile methods will work in the absence of above-average people (Nerur et al. 2005). The process of implementing agile practices was highly supported by the agile consultant.

Requirements - Handling the Requirements in an Iterative Manner: It can be difficult to handle the requirements in an agile, iterative manner. In agile methods the requirements are gathered throughout the iterative lifecycle (Cockburn 2006), whereas in documentation-driven development (such as the Waterfall model) the requirements are gathered and specified in the beginning of the project (Pressman 2000). Complying with quality standards includes documenting the requirements and as the documentation of these have to be handled in a strict manner and the approval of the documents have to go through several people, changing the documents are very difficult. This case study showed that the software group was not able to handle their requirements in a highly iterative manner, even though they experienced problems with the requirements specified. The reason was the very strict procedures for changing the requirements documents; especially those placed high in the document hierarchy. But, they did push one rewriting of the requirements specifications through late in the process.

Testing - Implementing a Test-Driven Process: It can be difficult to implement a test-driven process in a documentation-driven setting. Agile software methods suggest that test cases are written before the software and that all parts of an increment are tested within the iteration. These executable test cases define the requirements, while documentation-driven methods rely on documented test plans and procedures (Boehm, Turner 2003b). In this case study the software group had problems implementing a test-driven process. The software group did not (despite several attempts) succeeded in implementing agile testing practices in their software process. Much testing was postponed until the late stages of the project and in order to comply with the quality standards, much effort was put on creating test plans and procedures. Postponing the tests contributed to the releases of unstable increments to system tests, which returned many errors. Dealing with errors found at the system test run by the system engineering group implied a heavy authorization process, while errors found at tests within the software group could be handled in a much lighter way.

Customer-Relations - Implementing a Customer-Driven Process: For a software group developing software for a device aimed for the market it can be difficult to implement a customer-driven process. A close customer contact and a customer-driven process are advocated by the agile methods, in order to ensure that the end-product meets the needs and expectations of the customer (Boehm, Turner 2003b). Scrum operates with the role of the product owner; a person who is responsible for evaluating the system and hereby makes sure that the product fills the expectations of the customer. The product owner is also responsible for taking decisions about the system requirements (Schwaber, Beedle 2001). The documentation-driven software methods only involve the customer in the beginning of the product when creating the requirements (Boehm, Turner 2003b). The software group in this case study experienced a need for a product owner to (re)define the requirements; creating a relation to the customer was in this case study impossible as no users were affiliated to the project and as the management (driven by the quality standards and not by agile principles) did not prioritize this role. Instead this process had a strong documentation-driven character, even though the software group assigned the product owner role to software project manager; he was not able to make the final decisions for the requirements. As, the software group experienced troubles connected to the missing product owner, this was seen as a main weakness of the process.

CONCLUSION

This paper is a study of the software development process of a software group in a pharmaceutical company developing a highly safety-critical software system for drug dispensing. The software and the device in which it was embedded were therefore to be compliant with several process standards, for example the FDA standard. The purpose of this paper was to understand the possibility of implementing a software development process, which meshes the agile and the documentation-driven methods and the challenges of doing this. To do this, the analysis is based on a framework developed from the home grounds (Boehm 2002) of the agile and the documentation-driven methods.

The software development process presented in this paper meshes the practices of the agile software development method Scrum with a documentation-driven method complying with the FDA requirements. The software group was agile as they ran time-boxed iterations of 2 weeks, including a planning meeting and a retrospective. The control of the tasks and the progress of the iterations were supported by the use of scrum boards. The developers were divided into small self-organizing teams; within these teams they coordinated through daily stand-up meetings, while the scrum master facilitated the coordination between the two teams. The software process was however also documentation-driven as the requirements were fixed early in the project and there was no customer contact or validation of the interpretation of the requirements before implementation.

This case study supports the previous literature and proves that it is possible to implement a software development process, which meshes the agile and the documentation-driven methods. It shows that it is possible to mesh the project characteristics: planning and control, communication and knowledge sharing and environment. It is possible to develop the software in iterations within a long-term plan consisting of long-term milestones; but these iterations will be affected by the milestones as these are guiding the focus of the project. It is possible and advantageous to implement an agile, personalized communication and knowledge strategy, by it is still necessary to document the process as well. An informative workspace including for example scrum boards increases the visibility of the progress. It was however also proven difficult to mesh the project characteristics: developers, requirements, testing and customer-relations. For documentation-driven developers it can be very hard to change to an agile mindset in which the tasks need to broken down to fit in the iterations and in which knowledge sharing is primarily face-to-face. It is difficult to handle the software requirements in an iterative manner, as documentation of these is required and as the documentation goes through a strict approval process. Implementing a test-driven and a customer-driven process also requires priority from management, for example implementing a functioning product owner role has proven difficult without priority from management.

REFERENCES

Alegria, J.A.H. and Bastarrica, M.C. (2006). Implementing CMMI using a Combination of Agile Methods. *CLEI ELECTRONIC JOURNAL,* (9:1): 1-15.

Anderson, D.J. (2005). Stretching agile to fit CMMI level 3. *In: Proceedings of the Agile Development Conference,* IEEE computer society, Denver, p. 193.

Avison, D.E. and Fitzgerald, G. (2003). Where now for development methodologies? *Communications of the ACM,* (46:1): 78-82.

Baker, S.W. (2005). Formalizing agility: An agile organization's journey toward CMMI accreditation. *In: Proceedings of the Agile Development Conference,* IEEE Computer Society, Denver, CO, USA, p. 185.

Beck, K. and Andres, C. (2004). *Extreme programming explained: Embrace change,* Addison-Wesley Professional, USA.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J. & Thomas, D. , (2001), *Manifesto for Agile Software Development,* . Available: http://agilemanifesto.org/ [2011].

Boehm, B. (2002). Get ready for agile methods, with care. *Computer,* (35:1): 64-69.

Boehm, B. and Turner, R. (2005). Management challenges to implementing agile processes in traditional development organizations. *IEEE Software,* (22:5): 30-39.

Boehm, B. and Turner, R. (2003a). Observations on balancing discipline and agility. *In: Proceedings of the Agile Development Conference,* IEEE Computer Society, Salt Lake City, Utah, USA, p. 32-39.

Boehm, B. and Turner, R. (2003b). Using risk to balance agile and plan-driven methods. *Computer,* (36:6): 57-66.

Bos, E. and Vriens, C. (2004). An agile CMM. *In: proceedings of 4th Conference on Extreme Programming and Agile Methods-XP/Agile Universe,* Springer, Calgary, Canada, p. 129.

Chrissis, M.B., Konrad, M. and Shrum, S. (2003). *CMMI guidelines for process integration and product improvement,* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Cockburn, A. (2006). *Agile software development: The cooperative game,* Addison-Wesley Professional, Boston, USA.

Cockburn, A. and Highsmith, J. (2002). Agile software development, the people factor. *Computer,* (34:11): 131-133.

Cohn, M. and Ford, D. (2003). Introducing an agile process to an organization [software development]. *Computer,* (36:6): 74-78.

Crawford, B., Castro, C. and Monfroy, E. (2006). Knowledge management in different software development approaches. *In: Advances in Information Systems, LNCS 4243,* Springer, Izmir, Turkey, p. 304.

Dahlberg, H., Ruiz, F.S. and Olsson, C.M. (2006). The role of extreme programming in a plan-driven organization. *In: The Transfer and Diffusion of Information Technology for Organizational Resilience: IFIP TC8 WG 8.6 International Working Conference,* Springer, Galway, Ireland, p. 291.

Elshafey, L.A. and Galal-Edeen, G.H. (2008). Combining CMMI and Agile Methods. *In: Proceedings on the 6th International Conference on Informatics and Systems,* Faculty of Computers and Information, Cairo, Egypt, p. 27.

Fitzgerald, B., Russo, N.L. and Stolterman, E. (2002). *Information systems development: Methods in action,* McGraw-Hill, New York, USA.

Fritzsche, M. and Keil, P. (2007). Agile Methods and CMMI: Compatibility or Conflict? *e-Informatica Software Engineering Journal,* (1:1): 9-26.

Galal-Edeen, G.H., Riad, A.M. and Seyam, M.S. (2007). Agility versus discipline: Is reconciliation possible? *In: International Conference on Computer Engineering & Systems, ICCES'07.* IEEE CNF, Cairo, Egypt, p. 331.

Heeager, L.T. and Nielsen, P.A. (2009). Agile Software Development and its Compatibility with a Document-Driven Approach? A Case Study. *In: Australasian Conference on Information Systems,* Melbourne, Australien, p. 205.

Heimdahl, M.P.E. (2007). Safety and software intensive systems: Challenges old and new. *In: Future of Software Engineering (FOSE '07),* IEEE Computer Society, Minneapolis, Minnesota, p. 137.

Hoyle, D. (2006). *ISO 9000 quality systems handbook,* Butterworth-Heinemann, Oxford.

Iversen, J.K., Nielsen, P.A. and Nørbjerg, J. (1998). Problem diagnosis software process improvement. *Information systems: Current Issues and Future Changes IFIP,* (30:66-81.

Jakobsen, C.R. and Johnson, K.A. (2008). Mature agile with a twist of CMMI. *In: Agile, 2008. AGILE'08. Conference,* IEEE Computer Society, Toronto, p. 212.

Kähkönen, T. and Abrahamsson, P. (2004). Achieving CMMI level 2 with enhanced extreme programming approach. *In: Proceedings of the 5th International Conference, PROFES 2004,* Springer, Kansai Science City, Japan, p. 378.

Larman, C. (2004). *Agile and iterative development: a manager's guide,* Addison-Wesley Professional, Boston.

Laurila, P. (2004). Are CMM and agile methods really compatible? *In: T-76.650 Seminar in software engineering,* Software Business and Engineering Institute, Helsinki, Finland, p. 1.

Lee, I., Pappas, G.J., Cleaveland, R., Hatcliff, J., Krogh, B.H., Lee, P., Rubin, H. and Sha, L. (2006). High-confidence medical device software and systems. *Computer,* (39:4): 33-38.

Mahanti, A. (2004). Challenges in enterprise adoption of agile methods - A survey. *Journal of Computing and Information technology,* (14:3): 197-206.

Manhart, P. and Schneider, K. (2004). Breaking the ice for agile development of embedded software: an industry experience report. *In: Proceedings of the 26th International Conference on Software Engineering,* IEEE Computer Society, Washington, DC, USA, p. 378.

McAvoy, J. and Butler, T. (2009). A Failure to Learn in a Software Development Team: The Unsuccessful Introduction of an Agile Method. *In: Proceedings of the 16th International Conference on Information Systems Development (ISD2007),* Springer, Cairn, Ireland, p. 1.

McMichael, B. and Lombardi, M. (2007). ISO 9001 and Agile development. *In: Proceedings of the AGILE 2007,* IEEE Computer Society, Washington DC, USA, p. 262.

Moe, N.B., Dingsøyr, T. and Dybå, T. (2010). A teamwork model for understanding an agile team: A case study of a Scrum project. *Information and Software Technology,* (52:5): 480-491.

Muhr, T. (1991). ATLAS/ti—A prototype for the support of text interpretation. *Qualitative Sociology,* (14:4): 349-371.

Myers, M.D. and Newman, M. (2007). The qualitative interview in IS research: Examining the craft. *Information and Organization,* (17:1): 2-26.

Nawrocki, J., Olek, L., Jasinski, M., Paliswiat, B., Walter, B., Pietrzak, B. and Godek, P. (2006). Balancing agility and discipline with xprince. *In: Rapid Integration of Software Engineering Techniques, Second International Workshop, RISE 2005,* Springer, Heraklion, Crete, Greece, p. 266.

Nerur, S., Mahapatra, R.K. and Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM,* (48:5): 73-78.

Paetsch, F., Eberlein, A. and Maurer, F. (2003). Requirements engineering and agile software development. *In: Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises,* Citeseer, Linz, Austria, p. 308.

Paulk, M.C. (2001). Extreme programming from a CMM perspective. *IEEE Software,* (18:6): 19-26.

Pikkarainen, M. (2009). Towards a Better Understanding of CMMI and Agile Integration-Multiple Case Study of Four Companies. *In: Product-Focused Software Process Improvement: 10th International Conference, PROFES 2009,* Springer, Oulu, Finland, p. 401.

Pikkarainen, M. and Mäntyniemi, A. (2006). An approach for using CMMI in agile software development assessments: experiences from three case studies. *In: SPICE 2006 conference,* Luxemburg, p. 4.

Pressman, R.S. (2000). *Software engineering - A practitioner's approach,* McGraw-Hill Publishing Company, England.

Reifer, D.J. (2003). XP and the CMM. *IEEE Software,* (20:3): 14-15.

Schwaber, K. and Beedle, M. (2001). *Agile software development with Scrum,* Prentice Hall, Upper Saddle River, New Jersey.

Sutherland, J., Jakobsen, C.R. and Johnson, K. (2007). Scrum and cmmi level 5: The magic potion for code warriors. *In: AGILE 2007,* IEEE Computer Society, Washington D.C., p. 272.

Theunissen, W.H., Kourie, D.G. and Watson, B.W. (2003). Standards and agile software development. *In: Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology,* South African Institute for Computer Scientists and Information Technologists, Republic of South Africa, Johannesburg, p. 178.

Vinekar, V., Slinkman, C.W. and Nerur, S. (2006). Can agile and traditional systems development approaches coexist? An ambidextrous view. *Information Systems Management,* (23:3): 31-42.

Vogel, D.A. (2006). Agile Methods: Most are not ready for prime time in medical device software design and development. *DesignFax Online,* 1-6.

Walsham, G. (1995). Interpretive case studies in IS research: nature and method. *European journal of information systems,* (4:2): 74-81.

Wright, G. (2003). Achieving ISO 9001 certification for an XP company. *In: Extreme Programming and Agile Methods, Third XP Agile Universe Conference,* Springer, New Orleans, LA, USA, p. 43.

Submitted for the Information Systems Journal

Barriers to adoption of agile development practices: a knowledge transfer framework

Lise Tordrup Heeager and Peter Axel Nielsen
Aalborg University, Denmark

Abstract. Agile practices to systems development are believed to depend largely on the developers' competences, experience and knowledge and to a lesser degree on formal development processes. This paper investigates the knowledge transfer and barriers to the transfer of agile development practices in a pharmaceutical organization. A framework of barriers to knowledge transfer of agile practices is developed and applied to a case study of knowledge transfer between two software development teams in the same organization where the aim was to transfer experience and knowledge about agile software practices. The framework provides a greater understanding of the barriers to knowledge transfer of agile practices. The findings extend existing knowledge management and knowledge transfer literature with an application to agile development practices and advocate a focus on the potential barriers hindering the stages of the knowledge transfer process. This has implications for the adoption and diffusion of agile development practices.

Keywords: knowledge transfer, agile development, adoption and diffusion.

INTRODUCTION

Information systems development has for more than a decade been concerned with a change of approaches and perspectives towards more agile approaches (Conboy 2009) and agile ideas and practices (Dybå, Dingsøyr 2008; Senapathi 2010). The adoption of these new ideas has not been as easy and smooth as initially anticipated (Abrahamsson et al. 2009). Studies reveal several possible reasons such as: lack of the time and resources required (Moe et al. 2010); slow reorientation of management (Moe et al. 2010); perceptions of agile approaches (Dybå, Dingsøyr 2008); organizational culture (Iivari, Iivari 2010); and situational characteristics (Krasteva et al. 2010). The research on adoption of agile development approaches is still scattered with no clear direction and even contradictory findings (Dybå, Dingsøyr 2008). The adoption of agile ideas by one team in a software company is only a first step in a more widespread diffusion where many teams in a company and even whole companies are applying agile approaches. In this paper we take the perspective of the wider adoption by many teams. We focus here on how experience with agile approaches achieved in one team in a company can be moved or transferred to another team in the same company.

There are a few studies addressing how to transfer experience with agile approaches. Learning an agile approach where knowledge of the approach resides outside a team may be hindered by lack of change management (McAvoy, Butler 2009) despite a commitment to change. There are several paths that adopters may take in their changing of the use of agile approaches and adopt these in practice (Smolander et al. 2011). A single study reports from a case where they have transferred practical knowledge of agile approaches between two merged companies by using selected parts of the agile method XP (Tellez-Morales 2009). Based on this starting point we then address how knowledge and experience is transferred from one development team to another development team. In particular we pose the research question of: Which barriers can prevent the transfer of knowledge of agile approaches from one team to another? No existing research addresses this.

We approach the research question from the theoretical framing of knowledge transfer and in particular we utilise and extend a framework of barriers to knowledge transfer. Identifying and transferring knowledge between teams is generally a powerful mechanism for improving the productivity of an organization and create a significant competitive advantage (Argote, Ingram 2000). Re-using past knowledge to improve future practice (Desouza et al. 2005) reduces rework, while repeating successful processes increases productivity and the likelihood of further success (Lindvall, Rus 2002).

Knowledge transfer is in general a difficult task (Chau, Maurer 2004) and many organizations have failed (Szulanski 2000). Recognizing potential barriers to knowledge transfer is an important step towards understanding how to overcome these difficulties (Riege 2005). We extend this framework with findings from the literature and we then apply it to a case study of the knowledge transfer of agile practices between two software teams within the same large company. One team (Team A) has successfully implemented an agile systems development process and integrated it with quality management standards including standards from the US Food and Drug Administration. The focus in the case study is on the knowledge activities to transfer knowledge from Team A and on the implementation of a similar agile process in another team (Team B).

The remaining paper is organized as follows. In section 2 the literature on adoption of agile software practices, knowledge transfer and the barriers is reviewed and the framework for barriers to knowledge transfer of agile practices is presented. In section 3 the research approach is described. In section 4 the analysis of the knowledge transfer process and the barriers in the case study is presented. Finally the findings are discussed in section 5 and concluded on in section 6.

THEORETICAL BACKGROUND

This section presents the theoretical background and the analysis framework developed on the basis of the literature on the barriers to adoption of agile practices and the literature on the barriers to knowledge transfer.

Adoption of agile software practices in a traditional software organization

Even though the adoption of agile software practices appear to be straight forward, introducing agile practices into an organization has in many cases proven to be a challenging task (Mahanti 2004), which requires a significant amount of time and resources (Moe et al. 2010). The difficulties arise due to the differences between the agile and traditional software development methods (Nerur et al. 2005). Shifting from traditional practices to agile practices requires major alterations to the work (Senapathi 2010). In a literature study (Heeager 2011) it was concluded that the sole most important differences between traditional and agile practices are: (a) the amount of documentation required, and (b) the way requirements are handled. Agile approaches do not support the degree of documentation demanded by quality assurance standards, which often are referred to as traditional software development methods (Boehm, Turner 2005). Additionally, the agile requirements are mostly user stories written by the customer, while the traditional methods rely on requirement specifications defined early in the process. As both agile (Theunissen et al. 2003) and traditional (Baker 2005) approaches are highly flexible and adaptable it is possible to overcome these challenges and several researchers conclude that it is possible to implement agile practices and principles in a traditional software development process even when it is compliant with a quality standard e.g., (Baker 2005; Boehm, Turner 2003).

Software organizations need a guide to the adoption (or rejection) of agile practices (Taylor et al. 2006). Organizations attempting to adopt agile practices need guidance from a framework or a process (Pikkarainen et al. 2005; Sidky et al. 2007). The existing software process improvement approaches are targeting traditional software development (Pikkarainen, Salo 2006). Pikkarainen et al. (2005) suggest an agile deployment framework, which purpose is to make agile principles and practices part of an assessment process. Sidky et al. (2007) propose an agile framework consisting of two levels of assessment: one at the project level and one at the organizational level.

Adoption of a method or approach involves changes to the activities, values and norms of a team. The introduction of agile practices can therefore be seen as a learning experience (McAvoy, Butler 2009). Identifying the challenges of adopting and diffusing of agile practices in several software teams in the same organization can therefore be viewed through the lens of the challenges of transferring knowledge among organizational units.

Knowledge transfer

Knowledge transfer in organizations is “the process through which one unit (e.g., group, department, or division) is affected by the experience of another” (Argote et al. 2000, p. 151). This is supported by Riege, who defines it as “the application of prior knowledge to new learning situations” (Riege 2007, p. 48). What manifests knowledge transfer are the changes in the knowledge or performance of the recipient unit. Thus, by measuring the changes in knowledge or performance the effects of the knowledge transfer process can be evaluated (Argote, Ingram 2000).

The terms ‘knowledge transfer’ and ‘knowledge sharing’ are often used interchangeably in the literature even though they are different (Liyanage et al. 2009). While knowledge sharing mainly refers to the exchange of knowledge between individuals, knowledge transfer also includes higher organizational levels, for example a group or a division (Argote and Ingram, 2000). Nonaka (1994) define knowledge sharing as a stage in the process of knowledge transfer. Liyanage et al. (2009) conclude that knowledge transfer is not only about exploiting resources, i.e., knowledge, but also about how to acquire and absorb it to make activities more efficient and effective.

The literature distinguishes between two types of knowledge: tacit knowledge and explicit knowledge. Tacit knowledge is deeply rooted in the actions, the experiences as well as the ideas, values and emotions of the individual, which makes it difficult to share with others (Desouza 2003b). Explicit knowledge is transmittable in a formal, systematic language (Nonaka 1994). This can be done through codification, i.e., using a people-to-documents approach, where the knowledge is extracted from one person, made independent of that person, and reused for various purposes (Hansen et al. 1999). The personalization approach is the opposite, in which knowledge is shared through people-to-people interactions and dialogue (Crawford et al. 2006). Far from all of an organization's tacit knowledge can be made explicit, and far from all explicit knowledge can be documented (Lindvall, Rus 2002).

Transferring experience from one project to another project is difficult in software development (Lyytinen, Robey 1999). Software development knowledge is highly tacit in nature (Desouza 2003a), but the traditional software development methods involve a variety of document-driven processes and activities (Lindvall, Rus 2002), which fosters a tendency to over-document (Chau, Maurer 2004) and discourage learning (Lyytinen, Robey 1999). While the traditional software development methods have a tendency to focus on explicit knowledge and a codification strategy, the agile software development methods focus on sharing of tacit knowledge and personalization strategy (Chau et al. 2003). Moreover, developers have many concerns to consider, they often work under time pressure, requirements for projects are often specialized, and the culture of the users is different (Lyytinen, Robey 1999).

Barriers to adoption of agile practices and knowledge transfer

Adoption of agile software methods and knowledge transfer are both difficult practices, which can be influenced by several challenges and barriers (Mahanti 2004; Pikkarainen et al. 2005). This section collects these barriers and presents the analysis framework through which we will understand the case. The barriers identified can be divided into seven categories in the framework, see table 1.

Barrier	Description
Individual skills	Individuals and their skills are central to the outcome of a knowledge transfer process.
Motivation and willingness	The individuals may be more or less motivated to receive knowledge.
Time and resources	Time is a significant factor as the process of identifying and transferring knowledge is very time-consuming.
Organizational culture	Culture is often seen as the key inhibitor of effective knowledge transfer. Culture also plays a main role when adopting agile software practices, as the entire organization is affected.
Management style	The adoption of agile processes in a traditional organisation requires a change in management styles.
Trust	Without trust people are unlikely to share knowledge.
Infrastructure	Integrating an IT system can support the knowledge transfer process.

Table 1. The framework of barriers to knowledge transfer of agile practices

Individual skills are central to the outcome of a knowledge transfer process, without the right skills, knowledge is not likely to be transferred. If the individual is not part of a social network, it is not possible to access the knowledge (O'Dell, Grayson 1998). A lack of network ties can result in ignorance on both ends of the transfer, as the individuals do not realize that someone else has the knowledge, which they require or that someone else is interested in the knowledge they possess (O'Dell, Grayson 1998). Differences, such as age, gender, educational levels and ethnic background between the participants can hinder the knowledge transfer process, as power issues and problems understanding each other may occur (Riege 2005; Tellez-Morales 2009). The skills and emotions of the individual play an important role. The communication skills, both verbal and written, are fundamental to effective and successful knowledge sharing and transfer (Riege 2005). Without the ability to communicate the point will not come across and the knowledge will not be transferred. A good coordination between the participants also advances the transfer process (Szulanski 2000). The absorptive capacity of the individual also varies and despite the best intension, these are a boundary for the knowledge transfer (Lyytinen, Robey 1999; Szulanski 2000).

The heavy reliance on the human factors in agile software development methods poses challenges to their adoption (Esfahani et al. 2010; Senapathi 2010). Little evidence is found, that agile methods will work in the absence of above-average people (Nerur et al. 2005). The skills of traditional developers are somewhat outdated; they have to change their mindset to fit the agile principles. Changing the mindsets of people is very difficult (Moe et al. 2010). A fear of change may also play a role, as developers might doubt, that they are able to acquire the required agile skills and cope with the new agile environment (Mahanti 2004). In a study by Moe et al. (2010) they conclude that the transition to an agile self-managing team is one of the most important challenges when adopting agile practices, as traditional developers are focused more on independent work-processes. In order not to make rash and poor decisions, an appropriate level of sceptism is needed when adopting agile practices, but on the other hand being too sceptical or even resist change can be a barrier (Cohn, Ford 2003). Based on an empirical study of seven organizations, Cohn and Ford (2003) found that most of the developers had an appropriate level of scepticism, but some of the developers did resist the change.

Motivation and willingness can also be a barrier. Individuals may be more or less motivated to receive knowledge from the outside. Lack of motivation may result in procrastination, passivity, feigned acceptance, sabotage, or maybe rejection of new knowledge (Szulanski 2000). Knowledge sharing activities can neither be supervised nor forced of people; instead, the individuals need to see a value of the knowledge in order to share their knowledge voluntarily (Riege 2005). Transferring knowledge also entails people to be able to discard old practices and sustain new ones (Szulanski 2000). Getting people to change their practices is difficult as they easily fall back on well-known routines; it requires a level of willingness from the individual (Gupta 2008).

As the process of identifying and transferring knowledge is very time-consuming (O'Dell, Grayson 1998), *time* becomes a significant factor. If you lack the time to identify other individuals or groups with specific knowledge, lack the time to share knowledge or lack the time to implement the knowledge in the practice the risk is that the knowledge transfer only succeed partly (Riege 2005). Information overload is linked to the time issue; it becomes a barrier because it is difficult to learn anything when there is so much to know in little time (Lyytinen, Robey 1999). A lack of *resources* to create sharing opportunities is also a potential barrier to a successful knowledge transfer (Riege 2005).

In the knowledge management literature, *culture* is often seen as the key inhibitor of effective knowledge transfer (McDermott, O'Dell 2001). It is important that the corporate culture and the organizational design support the knowledge transfer practices (Lyytinen, Robey 1999; Riege 2005). An organizational design in silos creates divisions which only focus on their own accomplishments and tend to safeguard knowledge (O'Dell, Grayson 1998). Knowledge flows which are restricted in some directions are also a potential organizational knowledge barrier (Riege 2005). Large organizational units are unmanageable and make it more difficult to enhance contact and transfer knowledge easily (Riege 2005). Lack of formal and informal spaces in which knowledge can be shared are also a potential barrier (Riege 2005). Introducing an agile process affects the entire organization (Svensson, Host 2005), the organisational culture needs to change from policy and procedure-based to that of freedom of development and management by team members (Misra et al. 2010). The inability to make this change is a potential barrier of the introduction of agile practices.

Without a knowledge management strategy and without integrating it as a part of the organization's strategic approach the knowledge transfer is likely to fail (Riege 2005). The organization need to prioritize knowledge retention of highly skilled staff (Riege 2005). Mangers must make sure to clearly communicate the benefits and values of the knowledge transferring practices, as lack of managerial direction can limit these (Riege 2005). Allowing and rewarding knowledge sharing and knowledge transfer is also important (O'Dell, Grayson 1998).

The adoption of agile processes in a traditional organisation requires a change in *management styles* (Boehm, Turner 2005; Dybå, Dingsøyr 2008). If the management style is not able to shift from command-and-control management to leadership-and-collaboration and shift from a documentation-based strategy to one of management of tacit knowledge (Misra et al. 2010), an important part of the agile principle will not be implemented.

Lack of trust can also be a barrier. Without trust people are unlikely to share knowledge. They are reluctant to give away any knowledge if they fear that the knowledge may be misused or taken unjust credit for and they are reluctant to accept knowledge if they lack trust in the credibility of the knowledge or the actors it originates from (Riege 2005). Individuals may safeguard their knowledge out of fear that sharing may reduce or jeopardise their job security (Michailova, Husted 2003).

Integrating an IT system can support the knowledge transfer process. Such a system however needs an adequate level of informational and technical *infrastructure* (including support and maintenance). It further needs to support the practices of the organization otherwise actors may be reluctant to use such a system (Riege 2005). Reluctance may also arise when people are not familiar with the system; communication about the IT-system, training and demonstration of the system is therefore important (Riege 2005).

RESEARCH APPROACH

This is a longitudinal and interpretive case study (Pettigrew 1990; Walsham 1995). The empirical data were collected over a period of one year (May 2010 to April 2011) in a large pharmaceutical company and specifically with a focus on two software teams (A and B) and the context that they are part of. Data were analysed and validated iteratively through two phases, each focusing on respectively software team A and B. Table 2 summarizes the research approach.

Phase	Focus	Data collection	Duration	Data documentation	Data analysis
1	Software team A	8 qualitative interviews	May 2010	Interviews and observations audio recorded and transcribed	Evaluation of software development process
		6 qualitative interviews	June 2010		
		Document study	June 2010		
		Observations of planning and stand up meetings	August 2010		
2	Software team B	1 qualitative interview	August 2010	Interviews audio recorded and transcribed	Evaluation of the knowledge transfer process
		5 qualitative interviews	October 2010		
		5 qualitative interviews	March 2011		
		Document study	April 2011		

Table 2. Summary of the research approach

The data were collected in Phase 1 based on an interview guide (Appendix A) focusing on how Team A develops software and on how the development process combines agile and traditional methods. The interviews were conducted through two iterations and included the process manager, the software architect, software testers and software developers.

The data were collected in Phase 2 based on an interview guide (Appendix B) focusing on how Team B has adopted the development practices from Team A and on evaluating the knowledge transfer process. First, an interview with the software coordinator and the coordinator between software and hardware was conducted. The second and third round included interviews with the two full time software developers, interviews with the process manager from Team A and follow up interviews with the software coordinator and the coordinator between software and hardware. The interview guides focused on the transfer process, which knowledge mechanisms were used, an evaluation of the outcome and on the barriers hindering the implementation of the transferred knowledge about the software development process.

All interviews were audio recorded, transcribed and coded using Atlas.ti V.6 (Muhr 1991). The analysis of Team A's practices mapped the agile and the traditional software development practices and the knowledge of Team A. The analysis was written up in a report and validated by the interviewees from Team A. The analysis of Team B mapped the specific barriers hindering the knowledge transfer process, after which the barriers were categorized according to the framework of knowledge transfer of agile practices (see previous section). The results were written up in a report and validated by software team B and the process manager of software team A.

THE ANALYSIS

This is a case study of knowledge transfer of agile practices between two software teams within the same pharmaceutical company. Both software teams were part of projects that developed medical devices with embedded software, which for the final devices entail compliance and subsequent approval by several agencies and their quality standards. The knowledge transfer process was established to pass on the experience of software development from Team A to Team B in order to diffuse this knowledge within the organization. Figure 1 depicts the knowledge transfer process schematically.

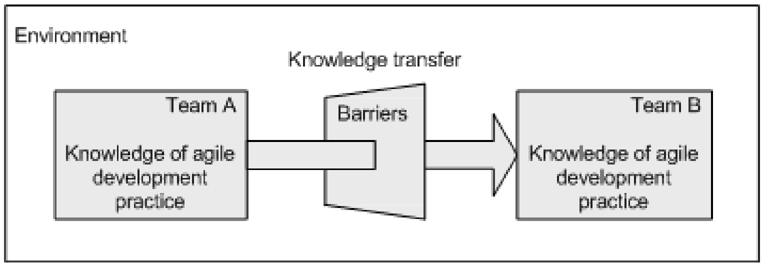

Environment
Knowledge transfer
Team A
Knowledge of agile development practice
Barriers
Team B
Knowledge of agile development practice

Figure 1. The knowledge transfer process. Within their organisational environment the knowledge of Team A is transferred to Team and the transfer is to some degree hindered by barriers.

The two teams are different in size and complexity; Team A's task is larger and more complex than that of Team B. While Team A consists of approximately 30 managers, developers and testers, software team B only consists of three developers, one of these in the role of the software coordinator. Project A has run for several years and is entering the stage of refining the product. So far project B has been focusing on developing prototypes. During the summer 2010, Team A changed its focus from the existing prototype to the overall design, developing two prototypes each focusing on a separate design. In late 2010 the team chose one prototype for further development, and afterwards the focus was on testing this prototype.

Team A had focused on developing, improving and documenting their software process. They have successfully implemented a software development process which includes agile practices, such as an iterative development process, self-organizing developer teams and coordination at daily stand-up meetings based on Scrum (Schwaber & Beedle 2002) and at the same time they were able to comply with FDA quality regulations and standards. They had gained experience in developing software architecture, conducting software test, peer reviewing and writing documentation. They have furthermore experimented with different tools to support the development process and been through the process of having these tools validated by various quality standards. The knowledge and experience of Team A is described in table 3.

Knowledge area		**Description**
Method (Scrum)	Iteration length	After experimenting with the iteration length the developers of Team A settled at 2 weeks sprints. It gives the developers a steady work rhythm, a dedicated focus and forces the developers to break the tasks down, which gives a transparency of the content of the tasks.
	Estimation of tasks	Software team A have experienced that estimating tasks are difficult, it takes practice and requires experienced developers.
	Product owner	The developers have experienced the need for a product owner to prioritize the tasks and to define the requirements. Assigning the role to a person who has the time and authority to fill the role has however proven difficult.
	Stand up meetings	The developers use daily stand up meetings to coordinate. These meetings ensure a shared focus and give an overview of the progress of the iteration.
	Scrum board	The software team uses a scrum board to visualize the progress of the iterations. Several advantages are associated with the board: the progress and status of the iteration is constantly visible and easy accessed and the board supports the discussions at the stand-up meetings.
Software architecture		The software team has found it necessary to define the overall architecture early in the project.
Software test		Much of the testing in project A is done late in the process, but they recommend to tests early and iteratively in order to discover the errors early. They also recommend close cooperation between the testers and the developers.
Documents		Due to the quality regulations the development process is highly regulated by documenting. Being the first software project in the organization The software team has designed the documents from scratch.
Peer reviews		The developers of software team A have found it advantageous to include the peer reviews as a part of the process. They do this by adding columns, to the scrum board, symbolizing the peer review.
Tools		The software team has experimented with different tools to support the development process which has resulted in a line of validated tools. They recommend using these tools from the beginning of the project.

Table 3. The knowledge of Team A

The knowledge and experience of Team A was deliberately transferred by several means: experience workshops, facilitation, consultancy and adaptation. All of the knowledge transfer initiatives were considered very useful by Team B. Table 4 gives an overview of the knowledge transfer activities and their content. Each knowledge transfer activity was identified and their contents determined based on the interviews; the description of each activity provided in the table is a summary of the descriptions given by the interviewees. The last column indicates the number of quotes marked in the transcriptions on each activity.

Activities	Contents	Description	Quotes
Experience workshop (21)	about Scrum	An agile consultant from software team A gave a two days introduction to Scrum to the developers of software team B. The content of the sessions was a mix of a theoretical description of Scrum and its terminology and a description of the Scrum process used by software team A.	11
	about tools	The process manager of software team A presented their line of tools and described the purpose of each tool. One of the employees in charge of the tools of software team A has furthermore held an experience workshop on their tool chain, explaining the advantages of each tool and given a quick demonstration of selected tools.	8
	about experiences of project A	In the beginning of the knowledge transfer process, the process manager gave a few informal presentations on the experiences of and challenges faced by software team A.	2
Facilitation (36)	of the Scrum process	For a month an agile consultant from software team A helped, software team B, introduce a Scrum process. In this period the team chose to run 1 week iterations in order to go through as many planning meetings, retrospectives and demonstrations as possible. The agile consultant was present during these meetings and some of the standup meetings. The facilitation process was considered very helpful and enlightening.	24
	of the software architecture	For 2 weeks the software architect of software team A helped define the software architecture of the system of software team B. He gave short presentations on software architecture and developed a suggestion to the software architecture. The purpose was to create a framework for the architecture and to advance the production of the software architecture document.	12
Consultancy (19)	of the software architecture	The software team has held 2 one-day meetings with an external consultant on software architecture. The first meeting took place in June 2010, while the second meeting took place in March 2011. In these meetings the consultant presented some architecture patterns. Software team A has received similar lessons by the same consultant.	19
Adaptation (38)	of processes and practices	The process manager of software team A and the software coordinator of software team B has since the beginning of the knowledge transfer process (spring 2010) scheduled weekly meetings. These meetings were however broken off for a couple of months. The meetings are informal and contain the subjects most urgent to the software coordinator. They discuss the problems currently faced by software team B in relation to the experiences gained at software team A.	15
	of documentation	Software team B has inherited the documents of software team A. These documents not only contain knowledge on how the software is developed in project A, but also represent knowledge on how to construct such documents.	19
	of tools	Employees of software team A have installed a tool chain at software team B.	4

Table 4. The knowledge transfer activities and the count of evidence in the coded empirical data.

Team B has gained much knowledge on Scrum, but have not been able to implement processes in their software practice yet to a level where they had gained their own experiences. They have also acquired specific knowledge on software test, software architecture and peer reviews, which become useful for the development of the software for the final device, but as they are still focusing on the software for the prototypes, these processes have not come into play. The tools have not been used either. Table 5 summarizes the knowledge of Team B.

Knowledge		Description
Method (Scrum)	Iteration length	Software team B has acknowledged the sprint length suggested by software team A, but they have not been able to implement sprints, yet. They have run a few test sprints of 1 week length.
	Estimation of tasks	During the test sprints the developers experienced trouble estimating the tasks and meeting the spring goal. Many interruptions in terms of ad hoch, support tasks also influenced their ability to deliver the planned tasks.
	Product owner	The role of the product owner has been assigned to the coordinator between the software, mechanics and hardware groups. This role is however not filled sufficiently mainly due to time issues.
	Stand up meetings	The software team B has not implemented a daily stand up meeting, yet. But they plan on implementing these meetings when introducing the sprints.
	Scrum board	The software team has designed a scrum board, but do not use it, as they do not run sprints, yet
Software architecture		So far the software team has not had the time to continue the work on the software architecture for the final device. They acknowledge the need for one.
Software test		Project B has little focus on test, as no formal tests have been made of the prototypes. The software team advocates the focus on early testing of the software for the final device, but the project management has a hard time understanding this need and do not allocate the resources (time and people).
Documents		The software team has obtained knowledge on how to build documents. Many of the documents have been processed.
Peer reviews		The software team has not done any peer reviews yet, as they have not developed any software for the final device.
Tools		Software team A has installed the tool train needed by software team B. Software team B has not used any of the tools yet and still do not know how to use them.

Table 5. The knowledge of Team B

A comparison of the knowledge of teams A and B reveals that the knowledge transfer process only has been partly successful, as Team B has not been able to integrate and use the knowledge in their own software practice. The remaining analysis will therefore focus on the barriers of the knowledge transfer process to reveal the underlying causes.

The barriers of the knowledge transfer of agile practices

The barriers to the knowledge transfer process were identified and presented in table 6.

General barrier	Specific barrier	Description	Quotes
Time and resources (59)	Focus on prototypes	The excessive focus on developing prototypes limits the time to transfer knowledge and adopt agile practices.	34
	Focus on overall design	The transfer of knowledge and the adoption of Scrum were put on hold, due to the focus on developing the overall design.	16
	Software developers not present during the knowledge transfer acts	The software developers were not present during all of the knowledge transfer initiatives. Due to the many urgent support tasks they were not able to allocate the time needed.	9
	Easier in the future	The time issues resulted in an attitude: "it will be easier in the future" among the developers.	5
Organizational culture (64)	The project management do not share organizational culture with software dev.	The project management has a background in mechanics and are having a hard time understanding software development.	33
	Disintegrated organizational cultures	The professional practice of the hardware and mechanics groups is different than the software group.	12
	Focus on mechanics	The deadlines of the entire project depend on the development of the mechanics.	19
Individual skills (15)	Not accustomed to Scrum	None of the software developers had any experience in using Scrum and were not trained in this method beforehand. As the organization is new at software development the software processes are not well-defined.	15
Motivation and Willingness (20)	Problems demotivate	The problems experienced during the sprints had a demotivating effect on the software developers, who otherwise were very motivated to implement Scrum.	17
	Management not present during the knowledge transfer acts	The project management was invited to the experience workshops but did not attend. The project management does not prioritize the knowledge transfer and the adoption of Scrum. This is a proof of their lack of motivation and willingness.	3
Management style (48)	Product owner	The product owner role has been assigned to the coordinator between the software, hardware and mechanics teams. As this is not his official role, he has difficulties allocating the time needed.	31
	Lean	The top management of the organization has introduced lean principles, which has entailed a display of a lean board, on which the deadlines for each project group is displayed. Detailed deadlines are requested, which leads to overlapping information on the lean board and the scrum board.	17

Table 6. The knowledge transfer barriers and their count of evidence in the empirical data.

Team B refers to lack of time as a primary reason why the Scrum practices have not been fully adopted, yet. At the same time of the knowledge transfer process the team has focused on the overall design and on developing prototypes of the system. These foci highly influence the daily practice. The software team was expected to support the hardware and mechanics teams with software test script and the majority of these are needed right away, hence difficult to plan in iterations. Supporting the prototypes is almost a full time job for all three software developers. The software for the final device has to comply with several medical quality standards; but the software for the prototype has not undergone any quality assurance, e.g., writing documentation, peer reviewing of the code and software testing. The development of the final software will therefore become more time consuming and the software team needs to start focusing on developing the software for the final device in order to finish within the final deadline for the overall project. But, the software team is left with very little time for the development of the software for the final device and little time to focus on the knowledge transfer and the adoption of new agile practices into their software process.

While focusing on the overall design and on developing the prototypes causes the time pressure, the specific barriers "software developers were not present during the knowledge transfer acts" and "easier in the future" is caused due to the lack of time. Due to the many urgent support tasks some of the developers were not present during the entire experience workshop and the facilitation of the agile process. An attitude "introducing Scrum will be easier in the future," arose among the software developers as they await a decrease in the time pressure.

As the organization and the project management have very limited experience in software development, they do not share the same organizational culture as the software developers. The project manager has a background in mechanics and has a hard time understanding the challenges of software development, as the process of developing mechanical parts is very different from the process of developing software. The disintegrated organizational cultures also appear as the mechanics and hardware groups lack the experience in software development. For example, they do not understand how the support tasks influence the software team and they do not define the tasks in advance in order for the software team to able to create these as backlog items and include these in the iterations. Getting the hardware and mechanics groups to follow the scrum principles is a great challenge for the software team. An excessive focus on the development of the mechanical parts of the system is furthermore an example of disintegrated organizational cultures in the organization and in the project.

None of the software developers had any experience in using Scrum in prior projects and were not trained in this method beforehand. Adopting Scrum is a difficult process and especially when being inexperienced. Due to their inexperience in running a Scrum process, the software team struggled with several problems when initiating the sprints facilitated by the agile consultant. One of the challenges was estimating the tasks and being able to meet the sprint goal. This problem mainly arose due to the large amount of ad hoch tasks related to the support of the prototype interrupting the sprints and the focus on the tasks prioritized by the product owner. The software developers are very motivated to learn about and adopt Scrum as their software development process, but this problem had a demotivating effect and was the main reason why the sprints were disrupted.

The project management is less motivated towards the adoption of Scrum. They were invited to the Scrum experience workshop and the Scrum meetings held in the sprints, but did not attend these, because of time issues. As the project management did not attend the knowledge transfer activities they do not understand practices, principles and advantages of Scrum. The lack of focus on the software process from the management has contributed to the lack of understand in the hardware and mechanics groups.

The roles of the scrum master and the product owner defined by the Scrum methodology require a change in management style. The project, including its management is not used to applying these roles and problems using these properly, especially the product owner, occur. The role of the product owner is to prioritize the tasks of the software team in order to create a focused attention to the most important tasks. As no customer is related to the project, the product owner role has been assigned to the coordinator between the software, hardware and mechanics groups, as he has an overview of the whole product and the needs of the three groups. This is however not an official role and difficulties allocating the time needed arise.

The top management of the organization has decided to introduce lean principles, which among other things has entailed a display of a lean board in all projects, on which the deadlines for each project team is displayed. The project manager requests detailed deadlines to be displayed on this board. Because of the level of details, information on the lean board and the scrum board is somewhat overlapping. The software coordinator has struggled (and still is) to keep the level of details down on the lean board, otherwise an overhead in updating the same information twice occurs. This issue arises as the project manager does not understand the principles of the scrum board and therefore seeks the same information on the lean board instead.

DISCUSSION

In summary, the analysis shows the transfer of knowledge of agile practices as a difficult undertaking. We shall first use figure 1 and the analysis to discuss the findings. The environment in which we have studied the case contains a strong imperative to include quality assurance processes (QA) in the software development process as required by the agency, in this case FDA, which has to approve the final product. It is additionally conditioning for the development teams that they are embedded in larger development projects that address the whole product and include teams for hardware development, mechanical development, and clinical development. Neither of these other teams nor the product project work in agile ways and have little knowledge of why that can be both necessary and useful for the software teams.

Table 7 provides a generalised overview of knowledge areas, transfer activities and barriers to knowledge transfer.

Team A Knowledge areas	Knowledge transfer activities	Barriers	Team B Knowledge areas
▬	→	▮	▬
Method (Scrum): iteration length, estimation of tasks, product owner, stand-up meetings, Scrum board Software architecture Test Peer review Documents Tools	Experience workshop (21) Facilitation (36) Consultancy (19) Adaptation (38)	Skills (15) Motivation (20) Time and resources (59) Organisational culture (38) Management style (48) Trust (0) Infrastructure (0)	Method (Scrum): iteration length, estimation of tasks, product owner, stand-up meetings, Scrum board Software architecture Test Peer review Documents Tools

Table 7: Knowledge transfer between team A and team B

This case study of knowledge transfer of agile software practices showed that, even though knowledge was transferred by several means, the knowledge transfer process was only partially successful. The knowledge of software team B covers the same knowledge areas, but is not the same as the knowledge of software team A and is not at the same level of understand and own experience. Software team B has retrieved knowledge of the experiences and software practice of software team A, but has not been able to implement the knowledge in their software practice, yet.

Software engineering has proven to be highly depended on tacit knowledge (Lyytinen, Robey 1999). The activities of software teams A and B are no exception. Adopting software practices and implementing these as a natural part of the software practice requires a transfer of tacit knowledge rooted in ones actions. As a deliberate strategy the knowledge transfer activities used in this case were designed to focus on transferring both tacit and explicit knowledge. The four kinds of activities in table 7 show this. The experience workshop was primarily conveying explicit knowledge, but did also include discussions between the consultant from team A and members of team B that pulled some of the consultant's more tacit knowledge. Facilitation, consultancy and adaptation had an a strong element of team A conveying their explicit knowledge, but in these activities there was a strong element of discussion and of team A trying to be helpful in solving the problems of team B. All in all, the knowledge transfer activities were many as seen from the evidence in table 4 and deliberately planned to overcome some of the difficulties of knowledge transfer known to the two teams.

However well planned the transfer was, there was several factors hindering or filtering the transfer. Software team B lacked experience in agile practices; while Software team A has spent years integrating the agile practices in their process. There was an identified need to try to transfer this experience from team A to team B. Knowledge transfer is acknowledged as a difficult undertaking and the barriers may be many. Knowledge management literature has acknowledged the importance of focusing on barriers (Szulanski 2000; Riege 2005) This claim is supported by this case study, in which the effects of knowledge transfer to a high degree is influenced by knowledge transfer barriers. Having identified the barriers hindering the transfer makes it easier to determine the right counter-measure to overcome or reduce the effects of the barriers.

The framework of barriers for knowledge transfer of agile software practices in table 1 has been applied as suggested in figure 1 with the results summarised in table 7. Having used the framework to analyse the knowledge transfer and its barriers it has provided us with a fuller description and understanding of each of the general barriers and how they specifically emerged and affected the results in this case. For each of the barriers we can now discuss their importance.

Organizational culture

The organizational culture has been identified as a potential barrier in both the knowledge transfer literature, e.g., (McDermott, O'Dell 2001) and in the literature of agile practices, e.g., (Misra et al. 2010). Svensson and Host (2005) propose that adopting an agile software development process affects the whole organization. Adopting an agile method requires radical changes which not only relates to the process of the software team, but requires radical changes in the project management and in the work flow of related project groups as well. A software group faces the challenge of distributing the processes and advantages of the agile method in order for the entire project to understand these and prioritize the time and resources needed. Organizational culture is a significant barrier identified in the case. This is specifically seen in the case where the project management and the software team do not share an organisational culture (33 quotes), disintegration between professional cultures (12 qoutes), and an overall focus on mechanics in the overall project (19 quotes). This supports the statement that adopting agile software practices affects the whole project. When a cultural change is needed one of the counter-measures proposed by Riege (2007) is to ensure an individual and collective understanding of the purpose, values and benefits. This underlines the importance of spreading the advantages of the knowledge transfer and the adoption of agile practices to the whole project.

Time and resources

A knowledge transfer process is more than likely to only succeed partially, if the necessary time to implement the knowledge in practice is lacking (Riege 2005). Adopting agile practices takes time - the adoption process of software team A has taken several years and so far the adoption process of software team B has been ongoing for almost a year without much progress. Time is a sign for prioritization. At the same time as team B prioritizes the adoption of the agile method Scrum is the development of prototypes. In the case it shows as a focus on prototypes (34 quotes) and focus on device design (16 quotes) as signs of prioritization. It also shows as not all software developers being present during planned knowledge transfer (9 quotes) and a naïve belief that there will be more time to transfer knowledge in the future (5 quotes). But, as the adoption of agile software practices affects the whole project, all teams in the project need to prioritize the adoption process in order to free the time needed. Getting the project management (and the other teams) to allocate the time needed for the knowledge transfer and adoption process is important.

Motivation and willingness

Knowledge transfer requires motivation and willingness from the people involved (Szulanski 2000). In this case study the software developers of software team B are very motivated and willing to receive knowledge and adopt the agile practices, but the project management and the other team in the software team's environment showed a lack of motivation. As the adoption of agile practices affects the whole project, and requires changes in work processes from not only the software team, but the other project groups as well, these people need to be motivated as well. Despite the high motivation of team B they were gradually demotivated when they realised that adopting an agile method like Scrum was not straight forward and posed several problems to the team (17 quotes).

Personal skills

The knowledge management literature underlines the importance of the individual. Agile software practices rely heavily on the individuals and require new and other skills than traditional software projects. Nerur (2005) advocates that agile software practices seem to require very skilled developers. The developers of software team B were not trained in agile methods before attempting to adopt the Scrum method to their software practice. The analysis shows that the fact that the developers experienced many problems when attempting the first sprints; some of these problems were related to the lack of personal skills of the developers (15 quotes). The case therefore supports the claim that adoption of agile practices require developers with agile skills. Teaching is one way to improve the skills of the developers. As the Scrum method depend on tacit knowledge and skills a personalization strategy (Hansen et al. 1999) is preferable. Another way is to hire additional developers with agile skills. Both of these strategies have been used in software team A and could be considered for team B as well.

Management style

Boehm and Turner (2005) have proposed that the adoption of an agile software method in a traditional organization requires a change in management style and therefore management style is a potential barrier of an adoption process. This case study supports the fact that management style is a potential barrier. The integration of the product owner role is an example of a change in management style, which in this case has proven difficult (31 quotes). It also turned out as a barrier that top management has committed to lean process development and for the software team lean and agile principles were not aligned (17 quotes). As a counter-measure Boehm and Turner (2005) propose educating stakeholders and translating agile software issues into management and customer languages. Software team B tried to include the management in the experience workshop, but without success. Their current strategy is to include them in the adoption process by holding meetings in which they present the progress and challenges of the adoption of Scrum. These meetings seem to serve this purpose of education.

CONCLUSIONS

This case study of knowledge transfer of agile practices confirms that adopting agile software practices is not always straight forward. The knowledge transfer can be affected by many potential barriers. In this paper we have developed a framework of barriers to knowledge transfer of agile practices. The framework has proven its relevance, as it provides a greater understanding of the barriers to knowledge transfer of agile practices and hereby gives a greater possibility to choose the better counter-measures to overcome the barriers.

The knowledge transfer process of the case study proved to be affected by most of the barriers of the framework. In the case study the organizational culture proved to have a great impact, as the adoption of the agile practices affects the whole project and not just the software team. The knowledge transfer suffered from lack of time, as the knowledge transfer and the adaptation of agile practices is not prioritized by the overall project. The software developers are highly motivated and willing to receive knowledge, but the lack of motivation from the overall project management restricts time that could be use on exchanging knowledge and implementing the knowledge in the software practice. It is therefore important for the software developers to get the project management and the other project groups motivated by conveying the advantages of the agile practices and hence committing to the changes imposed. The case study also showed that in order to implement the agile practices the knowledge transfer requires developers with agile skills. Projects with traditional developers should therefore offer training in agile practices. Management style has also proven to be a potential barrier to knowledge transfer of agile practices. Implementing the product owner role was one of the challenges met by the software team. Training of the project management therefore becomes necessary.

We thus suggest that for both practitioners and researchers working with diffusion of agile practices in an organization that our framework will be relevant and useful. Having identified the barriers it is important to choose the right counter-measure overcome the barriers. As this paper has not concerned itself with the development of strategies for counter-measures and the existing literature only contains little advice on how to deal with the barriers, more research on this subject is needed.

REFERENCES

Abrahamsson, P., Conboy, K. and Wang, X. (2009). ‘Lots done, more to do’: the current state of agile systems development research. *European Journal of Information Systems,* (18:4): 281-284.

Argote, L. and Ingram, P. (2000). Knowledge Transfer: A Basis for Competitive Advantage in Firms. *Organizational behavior and human decision processes,* (82:1): 150-169.

Argote, L., Ingram, P., Levine, J.M. and Moreland, R.L. (2000). Knowledge Transfer in Organizations: Learning from the Experience of Others. *Organizational behavior and human decision processes,* (82:1): 1-8.

Baker, S.W. (2005). Formalizing agility: an agile organization's journey toward CMMI accreditation. *In: Proceedings of the Agile Development Conference,* IEEE Computer Society, Denver, CO, USA, pp. 185.

Boehm, B. and Turner, R. (2005). Management challenges to implementing agile processes in traditional development organizations. *IEEE Software,* (22:5): 30-39.

Boehm, B. and Turner, R. (2003a). Observations on balancing discipline and agility. *In: Proceedings of the Agile Development Conference,* IEEE Computer Society, Salt Lake City, Utah, USA, pp. 32-39.

Boehm, B. and Turner, R. (2003b). Using risk to balance agile and plan-driven methods. *Computer,* (36:6): 57-66.

Chau, T. and Maurer, F. (2004). Knowledge sharing in agile software teams. *Logic versus approximation,* 173-183.

Chau, T., Maurer, F. and Melnik, G. (2003). Knowledge sharing: Agile methods vs. tayloristic methods. *In: Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on,* IEEE, Los Alamitos, USA, pp. 302.

Cohn, M. and Ford, D. (2003). Introducing an agile process to an organization [software development]. *Computer,* (36:6): 74-78.

Conboy, K., Wang, X. and Fitzgerald, B. (2009). Creativity in Agile Systems Development: A Literature Review. *Information Systems–Creativity and Innovation in Small and Medium-Sized Enterprises,* 122-134.

Crawford, B., Castro, C. and Monfroy, E. (2006). Knowledge Management in Different Software Development Approaches. *In: Advances in Information Systems, LNCS 4243,* Springer, Izmir, Turkey, pp. 304.

Desouza, K.C. (2003a). Barriers to effective use of knowledge management systems in software engineering. *Communications of the ACM,* (46:1): 99-101.

Desouza, K.C. (2003b). Facilitating tacit knowledge exchange. *Communications of the ACM,* (46:6): 85-88.

Desouza, K.C., Dingsøyr, T. and Awazu, Y. (2005). Experiences with conducting project postmortems: reports versus stories. *Software Process: Improvement and Practice,* (10:2): 203-215.

Dyba, T. and Dingsoyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology,* (50:9-10): 833-859.

Esfahani, C., Cabot, J. and Yu, E. (2010). Adopting agile methods: Can goal-oriented social modeling help? *In: Fourth International Conference on Research Challenges in Information Science (RCIS),* IEEE, Nice, France, pp. 223.

Gupta, K.S. (2008). A comparative analysis of knowledge sharing climate. *Knowledge and process management,* (15:3): 186-195.

Hansen, M.T., Nohria, N. and Tierney, T. (1999). What's your strategy for managing knowledge? *Harvard business review,* (77:2): 106-116.

Heeager, L.T. and Nielsen, P.A. (2009). Agile Software Development and its Compatibility with a Document-Driven Approach? A Case Study. *In: Australasian Conference on Information Systems,* Melbourne, Australien, pp. 205.

Iivari, J. and Iivari, N. (2010). 10 Organizational Culture and the Deployment of Agile MEthods: The Competing Values Model View. *Agile software development,* 203-222.

Krasteva, I., Ilieva, S. and Dimov, A. (2010). Experience-based approach for adoption of agile practices in software development projects. *In: Advanced Information Systems Engineering,* Springer, Hammamet, Tunisia, pp. 266.

Lindvall, M. and Rus, I. (2002). Knowledge management in software engineering. *IEEE Software,* (19:3): 26-38.

Liyanage, C., Elhag, T., Ballal, T. and Li, Q. (2009). Knowledge communication and translation–a knowledge transfer model. *Journal of Knowledge Management,* (13:3): 118-131.

Lyytinen, K. and Robey, D. (1999). Learning failure in information systems development. *Information Systems Journal,* (9:2): 85-101.

Mahanti, A. (2004). Challenges in enterprise adoption of agile methods-A survey. *Journal of Computing and Information technology,* (14:3): 197.

McAvoy, J. and Butler, T. (2009). A Failure to Learn in a Software Development Team: The Unsuccessful Introduction of an Agile Method. *Information Systems Development,* 1-13.

McDermott, R. and O'Dell, C. (2001). Overcoming cultural barriers to sharing knowledge. *Journal of knowledge management,* (5:1): 76-85.

Michailova, S. and Husted, K. (2003). Knowledge-sharing hostility in Russian firms. *California management review,* (45:3): 59-77.

Misra, S.C., Kumar, V. and Kumar, U. (2010). Identifying some critical changes required in adopting agile practices in traditional software development projects. *International Journal of Quality & Reliability Management,* (27:4): 451-474.

Moe, N.B., Dingsoyr, T. and Dyba, T. (2010). A teamwork model for understanding an agile team: A case study of a Scrum project. *Information and Software Technology,* (52:5): 480-491.

Muhr, T. (1991). ATLAS/ti—A prototype for the support of text interpretation. *Qualitative Sociology,* (14:4): 349-371.

Nerur, S., Mahapatra, R.K. and Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM,* (48:5): 78.

Nonaka, I. (1994). A dynamic theory of organizational knowledge creation. *Organization science,* (5:1): 14-37.

O'Dell, C. and Grayson, C.J. (1998). If only we knew what we know. *California management review,* (40:3): 154-174.

Pikkarainen, M. and Salo, O. (2006). A practical approach for deploying agile methods. *In: The procedures of the 7th International Conference on eXtreme Programming and Agile Processes in Software Engineering,* Springer, Oulo, Finland, pp. 213.

Pikkarainen, M., Salo, O. and Still, J. (2005). Deploying Agile Practices in Organizations: A Case Study. *In: EuroSPI 2005, LNCS 3792,* Springer, Berlin, pp. 16.

Riege, A. (2007). Actions to overcome knowledge transfer barriers in MNCs. *Journal of Knowledge Management,* (11:1): 48-67.

Riege, A. (2005). Three-dozen knowledge-sharing barriers managers must consider. *Journal of Knowledge Management,* (9:3): 18-35.

Senapathi, M. (2010). Adoption of Software Engineering Process Innovations: The Case of Agile Software Development Methodologies. *In: Agile Processes in Software Engineering and Extreme Programming, 11th International Conference, XP 2010,* Springer, Trondheim, Norway, pp. 226.

Sidky, A., Arthur, J. and Bohner, S. (2007). A disciplined approach to adopting agile practices: the agile adoption framework. *Innovations in Systems and Software Engineering,* (3:3): 203-216.

Smolander, K., Larsen, E.Å. and Päivärinta, T. (2011). Explaining Change Paths of Systems and Software Development Practices. *Information Systems Development,* 399-410.

Svensson, H. and Host, M. (2005). Introducing an agile process in a software maintenance and evolution organization. *In: Ninth European Conference on Software Maintenance and Reengineering, CSMR 2005.* IEEE, Manchester, United Kingdom, pp. 256.

Szulanski, G. (2000). The Process of Knowledge Transfer: A Diachronic Analysis of Stickiness* 1. *Organizational behavior and human decision processes,* (82:1): 9-27.

Taylor, P., Greer, D., Sage, P., Coleman, G., McDaid, K., Lawthers, I. and Corr, R. (2006). Applying an agility/discipline assessment for a small software organisation. *In: Product-Focused Software Process Improvement,* Springer, Amsterdam, Netherlands, pp. 290.

Tellez-Morales, G. (2009). XP Practices: A Successful Tool for Increasing and Transferring Practical Knowledge in Short-Life Software Development Projects. *In: Agile Processes in Software Engineering and Extreme Programming,* Springer, Pula, Sardinia, Italy, pp. 155.

Theunissen, W.H., Kourie, D.G. and Watson, B.W. (2003). Standards and agile software development. *In: Proceedings of the 2003 annual research conference of the South African institute of computer* *scientists and information technologists on Enablement through technology,* South African Institute for Computer Scientists and Information Technologists, Republic of South Africa, Johannesburg, pp. 178.

Walsham, G. (1995). Interpretive case studies in IS research: nature and method. *European journal of information systems,* (4:2): 74-81.