How software process improvement standards and agile methods co-exist in software organisations?

This thesis is presented for the degree of Master of Science of the University of Twente.

Author:

Ngoc Tuan Nguyen

n.t.nguyen-1@student.utwente.nl

Thesis MSc. Business Information Technology.

Enschede, August 2010.

Graduation commission:

Dr.Ir. Maya Daneva

MSc. Zornitza Racheva (PhD Associate)

Dr. Chintan Amrit

Dr. Klaas Sikkel

topicus

UNIVERSITEIT TWENTE.

Management summary

We are living in the world of extreme uncertainty. There is always an excess of changes, the unknown, and we often have to solve various life problems. In many cases, not only the solutions are unknown, but also the problems are unknown.

In the software industry, the situation is the same. Software projects are inherently chaotic and unpredictable, and as a result, cannot be managed by processes that are best suited to well-defined problem domains. Those processes are defined by traditional waterfall methodologies which are commonly used to date. Agile Software Development methodologies evolved out of the need to address the serious problem facing the software industry, and promise to help "software" people comfortable with chaos, ambiguity and the unknown.

Many software companies have successfully adopted agile methodologies. Moreover, the processes of adopting and adapting agile approaches take place concurrently with the process of getting more mature in those companies. There might have been two tracks of getting more mature: 1) get to higher level of agile maturity, in which agile practices and processes in a company become more disciplined and scalable; and 2) get to higher maturity by adapting or blending agile processes and practices with some standard maturity model such as CMM-SW or ISO 9001.

This research project first looks at literature to see the relationships between high level of maturity to the value and quality of software, as well as the value created to customers. We expected that high maturity level has positive impact on software quality, user satisfaction and value creation. The results drawn from literature review consolidate that, and more interestingly, they show that high process maturity significantly reduces waste as well.

As waste reduction, value creation and user satisfaction are believed to be the focus of agile methods, we next examine how agile practices and a maturity model can co-exist in an organisation and together bring more benefits to the organisation and its customers. The results show that the SPI standards really fit with agile methods, and that it is better if organisations can embrace both, since they benefit each other and bring more customer satisfaction as well as reducing waste and creating higher software quality and higher value creation.

After the literature study, some interesting questions arise, and the author further investigates them by means of a case study in Topicus, a software company based in Deventer, the Netherlands. The company has always been agile, and also achieved a certification for high maturity (SAS 70 with ITIL). However, while the company has gotten to more agile maturity, as well as gain more customer satisfaction and reputation, there has not been much combination and benefits from the certification to the software development side.

How to read this report?

We highly recommend that you read this report from beginning to the end. However, if at any point you want to look at specific important pieces of information, the following guide could be helpful:

- To get information on the research objective you can have a look at Chapter 2, especially sections 2.2. Research objective, 2.3. Problem statement and 2.4. Research questions.
- To get an overview of agile software development, agile methods and agile practices you can read Chapter 3. For the introduction to the most two popular agile methods XP and Scrum, please go to sections 3.5. Scrum and 3.6. XP.
- If you want to know about the definitions of all the main concepts used in this thesis, please read Section 4.1.
- The rest of Chapter 4 (sections 4.2, 4.3) is about the main findings from previously published studies about the issues that related to the research questions in Section 2.4.
- To get the results of the case study, please read Chapter 6, especially Section 6.4. Data analysis.
- To discuss the results of the case study as well as getting some recommendations for the case study organisation, please read Chapter 7.
- To get to know the conclusions, the limitations as well as the further research of the study in this master thesis, please read Chapter 8.

Preface

This thesis is the result of my final assignment for my study in the UT. The project takes me more than six months but the story about how I chose the agile software development area as my final research started long before. About one year ago, I contacted Dr. Maya Daneva for a research topic and she told me the term "*Agile Software Development*". I remember asking her exactly: "*Is that about coding?!*", and she answered that it is about software project management, and that Agile is the new way to manage the work that teams carry out to deliver a software product. Then I entered the "Agile world".

Since then, I have learned a lot more about agile philosophy as well as software development processes. The Agile world is so large as agile approaches are getting into the mainstream of software development. I believe agile is the most suitable approach in my home country, where most of the software companies are small and medium-sized organisations. In addition, many software companies in my country are either certified a CMM-SW or an ISO 9001, or pursuing one in order to be recognized as high mature organisations. Therefore, my study about the coexistence of software process improvement standard and agile methods in organisations would be really useful for me when I go back to my country to work.

I am indebted to a lot of people. Of course, the first person I would like to thank is Dr. Maya Daneva who introduced me to the field of research as well as introducing me to my other supervisors Msc. Zornitza Racheva and Dr. Chintan Amrit. Maya has supported me all the way, all the time. The second person I am very grateful to is Zornitza, who has been closely guiding me from the start of the project. It was amazing that she answered my emails at anytime. I am also thankful to Chintan for suggesting and revising my case study analysis as well as giving valuable comments on other parts of my research. Thank you all my supervisors for always being patient with my questions and for giving meaningful and useful feedback on my research.

I also would like to say thanks to Dr. Klaas Sikkel. One year ago, he and Maya, Zornitza together gave me some very helpful suggestions for starting my research in agile world, and for this master project, Klaas reviewed the questionnaire of my case study and gave me valuable comments. I am so grateful to Thijs Munsterman, Martin Krans, Johan te Winkel, and Niek Hoogma from Topicus for attending my interviews for the case study and answering my emails. Those Topicus people really impressed me so much, not only with the information they provided, but also with the way they welcomed me at Topicus as well as the way they responded to my questions.

As an international student, I never just simply take your all support for granted.

Ngoc Tuan Nguyen

Friday, July 23 2010,

Enschede.

Table of contents

List of tables

List of figures

Chapter 1: Introduction

This chapter aims to provide some background information regarding the research area in this master thesis. Moreover, the rationale behind the study and the related work done in previously published studies will also be discussed.

Numerous agile methods [45] have appeared in the last decade. Software development processes following agile methods (e.g., XP [2], Scrum [42]) and those following software process improvement (SPI) approaches (e.g., ISO 9001 [35], CMM [21] which define maturity levels) are normally considered contradictory [50]. Agile methods, on one hand, are light-weight methods and focus on fast delivery of valuable products. On the other hand, maturity standards have become heavy-weight and too much bureaucratic since they heavily focus on quality and documentation. CMM relies on institutionalization and documentation of process and methodologies, while Agile emphasizes interaction among workers and "working software over comprehensive documentation" (Agile Manifesto [1]).

Boehm and Turner [3], however, point out that organisations require both agility and discipline in order to be successful. Successful software development is challenged by the supplier's ability to manage complexity, technology innovation, and requirements change (or uncertainty at large). While management of complexity requires process discipline, management of change requires flexibility and adaptability [13]. Moreover, other researchers also conclude that while SPI standards guide organisations *what to do* in general terms, agile practices (e.g., XP) provide specific *how-to* information. Therefore, for the purpose of software success, the software engineering literature sources [3, 7, 8, 10, 11, 12, 13, 15, 37, 39, 43, 51] agree that agile methods and software processes at SPI-standards-compliant organisations can be complementary if cleverly combined.

With the advancement of agile methods and their penetration in the software industry, many people believe that agile methods and practices and maturity standards can co-exist in an organisation as a means to achieve excellence. One implication of that excellence is to develop the highest quality software in the shortest possible time.

Therefore, there is a need in fitting agile practices into high maturity organisations and also a need for organisations with agile nature to get more mature by following some standard. Normally organisations get certain level of maturity (i.e. achieving ISO 9001 certificate or some level of CMM/CMMI) and then burden a heavy-weight process of developing software, so they want to find out how they could fit agile practices in their existing world in order to accommodate changes, shorten time-to-market, or improve the communication among their staff. It is also the case that many software vendors and software solution providers have been following Scrum or XP but are experiencing market pressure to get ISO or CMMI certification, or are required by their customers to provide appropriate levels of documentation, or they have internal motivation for their own organisational improvement, lead to the demand of aligning with the ISO or CMM.

In this research, *process* refers to the description of phases in the product life-cycle through which the software is being produced, and *practices* are concrete activities and work-products that a method defines to be (regularly) used in the process.

1.1. Motivation

Nowadays there are more and more software companies pursuing a certification of software process maturity standards, and also a lot of firms adopting agile methods as an approach to increasing their software development efficiency. That might be because improving the development or working process is more focused, since it as important as product innovation. Although the two approaches can contribute to the same organisation's goals (e.g., developing higher quality software, gaining more customer satisfaction, and reducing waste), they are different approaches as introduced above. Therefore a study is needed on how the two ways can take advantages of each other to achieve the common goals.

1.2. Related work

There have been a number of publications focusing on the relationship between CMM/ISO and agile development [8] as well as literature sources discussing the application of CMMI/ ISO 9001 to agile software development [10, 12]. Previously published relevant studies by other researchers only discussed how organisations with certain level of maturity aligns or conforms to agile practices and the other way around, but do not examine thoroughly how agile practices and SPI standards in turn should adjust when adopted to organisations.

There are certain aspects in which this study is different from the others. On one hand, other studies [8, 9, 10, 11, 13, 15, 51] only discuss the compatibility of Agile methods and CMM/CMMI, or the application of a specific agile methods (XP or Scrum) in CMM or ISO 9001. Only a few examined the implementation of CMM or ISO 9001 in combination of agile methods, which are XP and Scrum (XP@Scrum [37, 47, 53]) [37]. Therefore, it is possible to make a conclusion that the issues on the "marriage" between agile methods and SPI standards have not been investigated in sufficient depth and breadth.

On the other hand, the study in this master thesis combines the issues and takes a step further toward a framework for adapting agile methodologies from a highly mature organisation. First, it investigates the relationship between higher process maturity and software quality, waste reduction, value creation and customer satisfaction. These four factors are the factors that Agile always values in its principles. Second, the study shows the similarity between the SPI standards (CMM/CMMI and ISO 9001 for software development) which are the manifestations of high maturity. Third, this study investigates how the combination of agile methods (both technical and managerial perspective) can be implemented in highly mature organisations. This investigation is supported by the benefits that those organisations can get when adapting agile practices.

Chapter 2: Problem definition

The objective of this chapter is to define the problem as well as introducing the way to solve the problem. Below, the following issues will be discussed: the boundaries of the research project in this master thesis (Section 2.1), the research objective (Section 2.2), the problem statement (Section 2.3), the research questions (Section 2.4) and the research methods that are used (Section 2.5).

2.1. Project boundaries

This master project research focuses only on common SPI standards such as ISO 9001 and CMMI (and its predecessors), as well as the most popular agile software development (or application development) and management methods which are Scrum and XP. In our study, the literature sources are publications including papers in scientific libraries (Scopus) and books by prominent practitioners on agile software development.

The research context is that of organisations willing to blend agile methods with SPI standards. However, this project also includes empirical research that is a case study carried out in a Dutch agile software company, Topicus. Therefore, we will narrow the discussion on the marriage of agile and SPI-standard based certification as it applies to the particular contextual settings of this company.

2.2. Research objective

This research aims to provide a better understanding about the relationship between SPI standards and agile practices as well as to identify the possible ways for improvement that co-existence in some specific settings, and to recommend some guidelines for their adoption to each other. It is the case that the majority of people (even if they adopt CMM or ISO) agree that in most cases they understand what need to be improved, but they need more guidance about how to improve it.

The objective of this research is to take the best of SPI standards and the agile methodologies and combine them. To do that, this research will show how the most widely used agile methods (e.g., XP and Scrum) address the areas of SPI standards (e.g., CMM). This is useful for those organisations that have their plan-driven process based on an SPI standard model and are planning to improve its processes towards agility.

2.3. Problem statement

Many researchers and leaders in software industry agree that agile practices like XP and Scrum and the process disciplines like CMM and ISO 9001 are theoretically compatible. However, putting agile methods to work in such disciplinary environments is often difficult. In the other way around, it is not easy as well for organisations with "agile culture" to obey discipline requirements and get an ISO or CMMI certification. Guidance on how to take advantage of existing best practices in both approaches is needed while adopting each other.

2.4. Research questions

In order to achieve the research goal, two main questions need to be answered:

2.4.1. Research question 1 (RQ1)

What are the relationships between maturity level and software quality, waste reduction, value creation and user satisfaction?

In this question, the term "maturity level" refers to the degree of traditional maturity alone, which is usually defined by a CMM-SW [21] or ISO 9001 [35] requirements and without considering the adoption of agile methods. Software quality, waste reduction, value creation, and user satisfaction are examined in the software development as a whole.

As the IT industry is growing very fast due to rapid innovations, there has been an intense competition among IT firms at large as well as among software development companies. Therefore, making high quality software products, delivering on-time and on-budget (or low-costs) has always been crucial for software firms.

In the context of this research, *software quality* is the focus of both SPI standards and agile methods, but the concepts of *waste reduction*, *value creation,* and *user satisfaction* are of the main benefits that agile methods promise [1, 3, 60, 64].

We make the note that RQ1 means the author of this research expects that the higher level of maturity, the higher quality of software delivered and the less waste reduction exists. Moreover, as value creation and user satisfaction depend on context and are client-dependent, this research will also examine in respect to what aspects value creation and client satisfaction increase or decrease in conjunction with the growth of maturity.

2.4.2. Research question 2 (RQ2)

What is the relationship between process maturity models and agile software development methods?

For this question, there are three sub-questions:

i) Does process maturity models fit with agile methods

ii) How can agile methods benefit from higher level of maturity of an organisation?

iii) How can agile methods be adapted in organisations that have high level of maturity?

We make the note that the two main research questions (RQ1 and RQ2) are inter-related. They are all about how maturity level and agile methods can take advantages of each other strength as well as minimizing the disadvantage of each approach on the adoption of the other.

For the purpose of clarity, the research questions are summarized into the research model depicted in Figure 2.1.

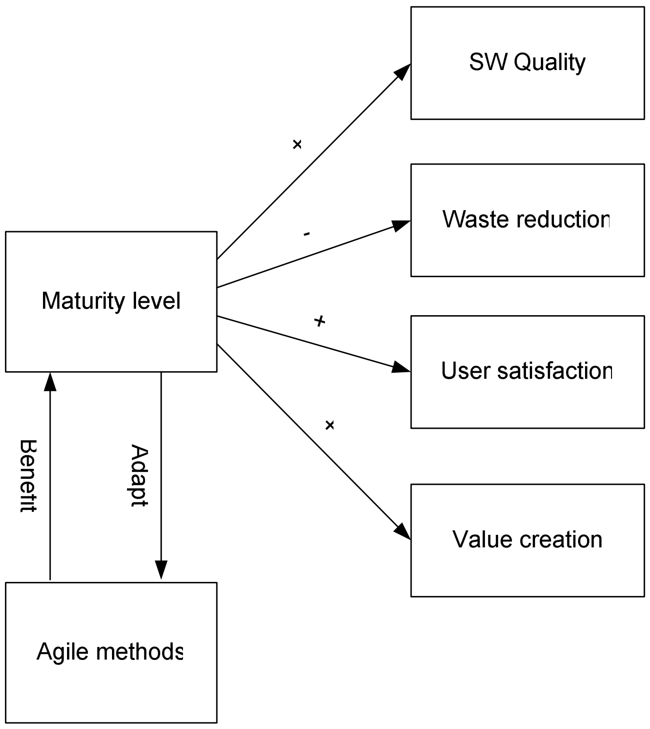

Figure 2.1. Research model.

2.5. Research method

In this master thesis we conduct a *qualitative research* [14] in order to obtain a good understanding on the problem that is being investigated. In this approach to empirical research, the focus is to study the objects and concepts in their own environment.

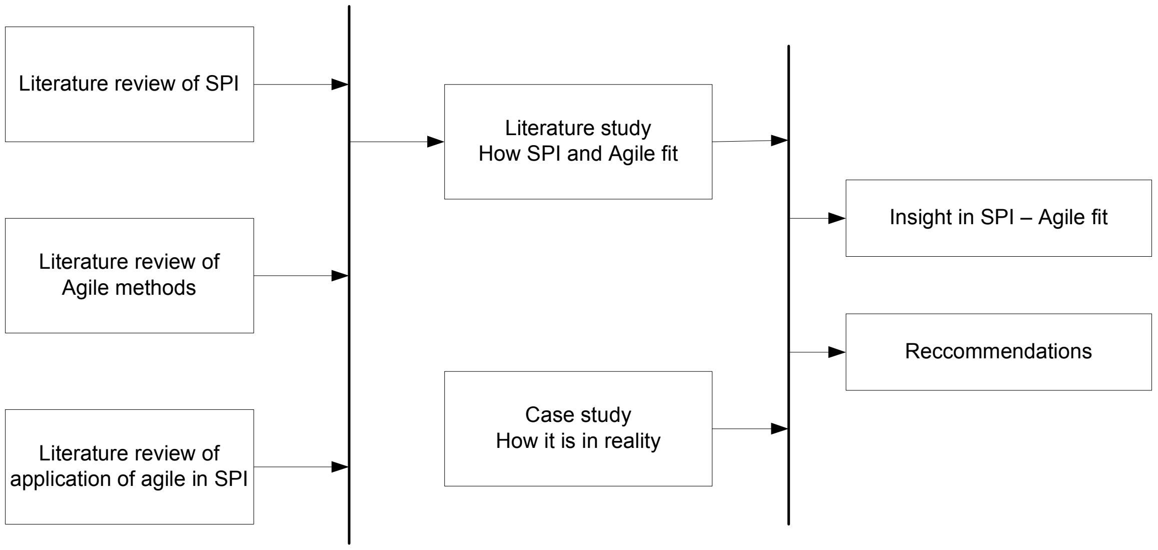
Literature review of SPI
Literature review of Agile methods
Literature review of application of agile in SPI
Literature study How SPI and Agile fit
Case study How it is in reality
Insight in SPI – Agile fit
Reccommendations

Figure 2.2. Research approach.

As shown in the Figure 2.2, the main research methods to be used in this project are literature study and case study. The literature study on how SPI and Agile fit together is served by three literature reviews as the inputs.

Firstly, for *Literature review of SPI*, the traditional way of developing software and the SPI standards such as CMM-SW and ISO 9001 are studied. Their main principles, patterns and practices are discussed to show how and where they best work.

Secondly, this research looks at Agile as a toolbox in the *Literature review of Agile methods*, where Agile as a philosophy, as a value system, and as a software development paradigm, is discussed. Furthermore, the most popular agile practices or methods such as XP and Scrum, which have chosen elements from the general agile toolbox, are deeper dug into. Those practices are what practitioners really do nowadays. Similar to the review on SPI, the review on Agile also shows how and where agile practices work best.

Lastly, and closer to the main literature study part, is the review on *the application of Agile in SPI*. This part will review from existing studies, research and publications on how agile methods or practices can be matched to SPI processes and practices. That is, about the matches among CMM-SW, ISO 9001 and Scrum, XP.

Then, the conclusions on how SPI and Agile fit together are formed from the literature reviews above. Some ideas from the author of this research are also expressed. This shapes Chapter 4 (Literature background).

Next, in this project we take one specific organisation as the site of a case study about how practitioners fit their existing agile philosophy, values, and development approach to the goal of achieving an SPI-standard based certification. The case study, Topicus is a Dutch middle-sized company adopting agile and getting more mature.

Drawing on the results from our literature study and our case study, this research comes up with the answers of the research questions and with guidelines and recommendations for companies in certain contexts to deal with achieving higher maturity with agile, and shows whether the research objectives are met or not.

2.5.1. Literature review

The systematic review [49] will be conducted from literature on SPI standards, their guidelines, agile methods, and agile practices as well as literature on the relationships between SPI standards and agile methods. For agile software development methods and practices, since there have been some other reviews done by well-known researchers ([45, 46]), we will not do another systematic review but draw on their results. The main ideas, concepts and results are discussed in some parts of the Chapter 3 (Agile development) and in Chapter 4 (Literature background).

2.5.2. Case study

The case study will be conducted according to Yin [14] which consists of the following phases:

- Design: in this research, it is a single exploratory case study
- Implementation: the designed case study is developed to answer the (sub) research questions. A questionnaire is made, together with some documentations on the case will be collected. Moreover, there can be both electronic (email) interviews and face-to-face interviews.
- Analyse the data gathered
- Draw conclusions and recommendations

More details about the case study will be discussed in Chapter 6 of this report.

We choose the case study research method because of the following reasons. Firstly, a case study can extend our experience and our understanding of the issue what is already studied through previously published research. Before carrying out the case study, a literature review will be conducted that leads to some refined, insightful questions about the issue. Secondly, a case study emphasizes detailed contextual analysis of a set of constructs and their relationships. The object of the case study in this research is an organisation or a group of people and they are likely to be connected cultural, social, and personal issues. This provides a wide range of data for the researcher to investigate the problem in depth. And thirdly, since this study is in a preliminary stage (when only a few phenomena are observed), it mainly explores the phenomenon of the adaptation of agile practices and processes to SPI standards in real-life situations, and an *exploratory case study* is a suitable tool. Yin [14, p. 13] defines the case study research method as an empirical inquiry that investigates a contemporary phenomenon within its real-life context, when the boundaries between phenomenon and context are not clearly evident, and in which multiple sources of evidence are used.

Chapter 3: Agile development

This chapter will first give an introduction into agile software development, which consists of agile values, agile principles, and agile practices based on those values and principles. Next, some results are drawn from existing reviews ([45, 46]) about agile methods, their processes and practices. Lastly, the two most widely used agile methods (eXtreme Programming and Scrum) will be discussed.

3.1. Introduction to agile development

Agile – denoting "the quality of being agile; readiness for motion; nimbleness, activity, dexterity in motion" [45] – software development methods are attempting to offer an answer to the eager business community asking for lighter weight along with faster and nimbler software development processes. This is especially the case with the rapidly growing and volatile Internet software industry as well as for the emerging mobile application environment.

Agile development is a way of organizing the development process, emphasizing direct and frequent communication – preferably face-to-face, frequent deliveries of working software increments, short iterations, active customer engagement throughout the whole development life-cycle and change responsiveness rather than change avoidance [12]. Thus, agile software development recognizes that software development is inherently a type of product development and therefore a learning process. It is iterative, explorative and designed to facilitate learning as quickly and efficiently as possible. Two of the most significant characteristics of agile approaches are: 1) they can handle unstable requirements throughout the development cycle; and 2) they deliver products in shorter time-frames and under budget constraints when compared with traditional development methods.

An agile approach can be seen as a contrast to waterfall-like processes [42, 52, 66] which pay attention to thorough and detailed planning and design upfront and consecutive plan conformance. The waterfall model is the oldest and the most mature software development model [52]. In practice, the waterfall development model can be followed in a linear way, and an iteration in an agile method can also be treated as a miniature waterfall lifecycle.

Table 3.1 (on next page) summarizes some of the differences between waterfall and agile projects.

Usage of agile approaches

Agile approaches have been widely employed in a domain of low cost of failure or linear incremental cost of failure [68]. Examples within this domain include web-based applications, mobile applications [45], Internet commerce, social networking, games development, and even some areas in government, finance and banking software development.

Criteria	Waterfall	Agile
Product/scope	An often bloated product that is still missing features (i.e., rejected change requests or de-scoped to meet deadlines)	The best possible product according to customers own prioritization, incorporating learning from actual use (revolves with the increments)
Schedule/time	Deadlines are usually missed, and it is unlikely for a project to deliver early.	Very high probability of meeting fixed date commitments; can often deliver early with the highest value.
Quality	Defects must be tested extensively and expensively.	Quality is built in, and is the key to productivity (writing tests before writing code).
Return/value creation	Revenue earning and value creation are delayed until the lowest priority features are implemented and delivered.	Value is generated early, as soon as the minimum highest prioritized features are delivered. Greater return on investment.
Relationship to the customer	Contractual	Collaborative

Table 3.1. The differences between Waterfall and Agile projects

3.2. Agile practices

Experiencing a project with no practices to guide its execution is a nightmare. Firstly, the lack of effective practices leads to unpredictability, repeated error, waste effort, and so on. Moreover, customers (or clients) are disappointed by slipping schedules, growing budgets, and poor quality. Also, developers are disheartened by working ever longer hours to produce ever worse software. This part discusses the principles behind all agile practices contemporarily used.

3.2.1. The Manifesto for Agile Software Development

The Manifesto for Agile Software development was created in 2001 by Agile Alliance – a group of experts in software industry. The Manifesto [1] says:

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

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation

- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

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

Firstly, by valuing **individuals and interactions over processes and tools**, the manifesto confirms that people are the most important ingredient of success. A good process will not help the project deliver the truly satisfying product if the team does not have strong enough players (who work well with others). Moreover, it is the case that even a group of skillful people can fail badly if they do not work effectively as a team [40]. Furthermore, the valuation suggests that the right tools (e.g., editors, compliers, IDEs (Integrated Development Environment), source-code control systems, etc.), or the development environment at large, can be very important to success but they should not be overemphasized. Building the team is more important than building the environment.

Secondly, **working software is considered more important than comprehensive documentation**. This mean a development team should concentrate on developing valuable software for customer, and "produce no document unless its need is immediate and significant" [40]. *Working software* is fully tested software that is ready for release, in the context of agile development it is also a package of high-value prioritized functions and features. However, it is obvious that software without documentation is a disaster. The team needs to produce human-readable documents that describe the system and the rationale for their design decisions. Documents are needed for transferring "knowledge" among the members of the team, between the team and customer, or for maintenance or reuse in the future (or the knowledge management of the organisation at large). In the cases documentation is needed, however, it should be also short and only the most important features or the highest-level structures in the system are included.

Thirdly, the manifesto values **customer collaboration over contract negotiation**. A normal contract specifies the requirements, schedule, and cost of a project. In most cases, those terms are not appropriate even long before the project is finished. For software product, mostly customer do not know exactly all what they want (the requirements) in the beginning. They sometimes cannot even articulate what they need the software to have. So the best contracts are those that specify the way the development team and the customer would work together. To be successful, the project should involve customer feedback frequently by closely working between customer and the development team.

And lastly, **responding to change is valued over following a plan**. It often the case that the ability to respond to change that determines a software project successful or not. Therefore, when building a plan, we should make sure that the plan is flexible enough to be ready for changes in the environment (business and technology). Due to the uncertainty in reality, a software project cannot be planned very far into the future. Firstly, the business environment is likely to change, causing the requirements to shift and the schedule to adapt. Secondly, customers are likely to alter their requirements once they see the working software (*"I will know when I see it" (IWKWISI)* seems not to be an uncommon customer's statement). That is why a good planning strategy should be only detailed for the next two or three weeks, and the time beyond that should only be planned roughly, so that we can better respond to change.

3.2.2. Principles

There are twelve principles behind the Manifesto described above. The principles, which form agile concepts, are also characteristics that differentiate agile practices from a heavy-weight process [1, 2, 4, 40]:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a reference to the shorter time scale.
- Business people and developers must work together daily throughout the project.
- Build project around motivated individuals. Give them the environment and the support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility
- Simplicity – the art of maximizing the amount of work not done – is essential. The reason is change is inevitable, thus planning for the future functions is a waste of time (YAGNI – You Aren't Going to Need It).
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

So the Agility in software development means not only quick delivery of software products but also quick adaptation to changing requirements. Firstly, the agile approach is called into action when a project features incremental changes, particularly those that have not been included in initial requirement documents. Secondly, most agile methods attempt to minimize risks by developing software in short periods, called iterations (one to four weeks).

Each iteration includes tasks needed for releasing a working software in the end of the iteration which meets minimal highest priority requirements. Working software is the primary measure of progress. At the end of each iteration, the rest requirements are re-prioritized. Thus agile methods "embrace" change, focus on simplicity, and release often. Moreover, agile methods emphasize the face-to-face, real-time communication over written documents. Each development team also includes some people from clients (business side) to get feedback frequently. This means most agile teams include all people necessary to finish the software, which are developers, project managers, business analysts, actual (on-site) customer, and testers. All these members are typically co-located.

Tacit knowledge

As can be implied from the principles, agile methods rely on *tacit knowledge* [3, 56]. Tacit knowledge is highly personal. It is hard to formalize or document and, therefore, difficult to communicate to others [56]. Tacit knowledge is also deeply rooted in action and in individual's commitment to a specific context, and can only be learnt through observation, imitation, and practice. One of the obvious examples of tacit knowledge is technical skills – the kind of informal, hard-to-pin-down skills captured in the term "know-how" [74]. In a software project, agility is achieved by establishing and updating the project knowledge in the participants' heads rather than in documents (*explicit knowledge*) [3]. Explicit knowledge in terms of documentation and design should be kept to a minimum.

3.3. Agile Software Development

Agile software development (ASD) is a conceptual framework for undertaking software projects. It is defined as "evolutionary approaches which are collaborative and self-organizing in nature, producing high-quality systems that meets the changing needs of stakeholders in a cost effective and timely manner" [22]. There are a number of agile software development methodologies such as eXtremely Programming (XP, see more 3.6. eXtreme Programming), Crystal methods, Dynamic Systems Development Model (DSDM), Feature-Driven Development (FDD).

3.4. Agile Project Management

Agile project management (APM) approaches refer to "the work of energizing, empowering and enabling project teams to rapidly and reliably deliver customer value by engaging customer and continuously learning and adapting to their changing needs and environments" [22].

The best known and most used APM method (or the best APM practice) is Scrum (the list of companies using Scrum includes IBM, Microsoft, SAP, Philips, Google, and Yahoo [67, 79]), whose detailed flow is shown in Figure 3.1.

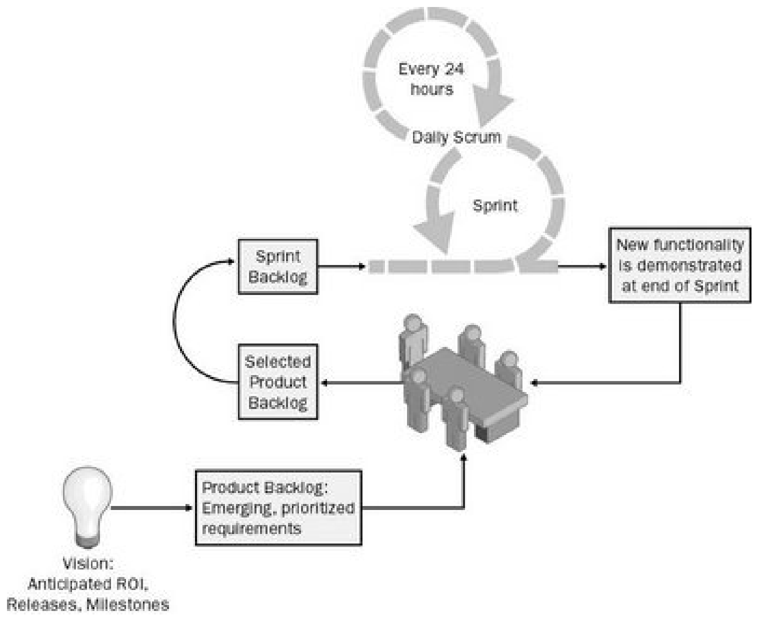
Every 24 hours
Daily Scrum
Sprint
Sprint Backlog
New functionality is demonstrated at end of Sprint
Selected Product Backlog
Product Backlog: Emerging, prioritized requirements
Vision: Anticipated ROI, Releases, Milestones

Figure 3.1. Scrum process overview [38].

APM practices more clearly separate from ASD practices in the challenge of scalability. For example, XP mostly works well with small-sized teams but Scum has evolved to fit any team size [68, 69, 79].

The line of the present research seems to focus much more on APM than on ASD. However, in this thesis, the author mainly considers agility as a philosophy, thus takes into consideration both ASD and APM practices in the same way. This means when saying "agile practice", the practice can be either software development practice or software management practice.

3.5. Scrum

As can be seen in Figure 3.1, Scrum is a light-weight process to manage and control development work. That is, it implements process control techniques. It is a team-based approach to iteratively, incrementally develop systems and products when requirements are rapidly changing.

Scrum is based on the concept that software development is not a defined process, but a "black box" (empirical) process with complex input/output transformations that may or may not be repeated under differing circumstances [42, 45]. It has a definite project management emphasis. Scrum is managed by Scrum Master, who can be considered as much a consultant or a coach as a manager. Moreover, there is *a fundamental 30-day development cycle (or a Sprint)*, which is preceded by a pre-Sprint activities and post-Sprint activities. A short (no more than 30 minutes) daily Scrum meeting allows the team to monitor status and communicate problems (Figure 3.1).

Project planning in Scrum is based on a Product Backlog [44], which contains functions and technology enhancements imagined or formed for the project. There are two meetings held, one is for deciding the features of the next Sprint and the other for planning out the work. In addition, a Sprint Goal is set which serves as a minimum success criterion for the Sprint and acts to keep the team focused on the broader picture rather than narrowly on the task at hand.

Therefore, Scrum offers a means of introducing agile methods into a traditionally disciplined environment. However, Scrum is not conceived as an independent method, but a complement to other methods like XP. Therefore, Scrum stresses management values and practices, but it does not include practices for the technical parts (requirements, design, and implementation) which are already complemented by other agile methods (see more in *Appendix B. Scrum*). Since Scrum's emphasis is on project management, it requires certain management practices and tools in its various phases to avoid the chaos caused by unpredictability and complexity [42]. Table 3.2 presents three lists of the different Scrum practices that have been adopted and adapted in reality and mentioned in previously published studies [44, 45, 65].

Schwaber & Sutherland [44]	**Abrahamsson et al [45]**	**Upender [65]**
Sprint	Sprint	Sprint
Daily Scrum	Daily Scrum meeting	Daily Scrum
Sprint planning meeting	Sprint planning meeting	Sprint planning
Sprint review	Sprint review meeting	Sprint review
Sprint retrospective	Not applied.	Not applied.
Release planning meeting	Effort estimation	Not explicitly mentioned.
Product backlog (User stories/use cases)	Product backlog	Product backlog
Sprint backlog	Sprint backlog	Sprint backlog (as a subset of product backlog)
Release burn-down	Not mentioned.	Burn-down chart
Sprint burn-down	Not mentioned.	Burn-down chart
Just-in-time planning at Daily Scrum	Not mentioned.	Just-in-time design

Table 3.2. Different Scrum practices (extracted from [44, 45, 65]).

It is possible to conclude from Table 3.2 that Scrum practices have been sustained and refined and that in each specific context or situation, the adoption of the practices may include some adaptation both in terms of the names and the content.

3.6. Extreme programming (XP)

Extreme Programming or XP was created for small and medium size software development teams where requirements are vague, change rapidly or are very critical. XP was designed in order to address the problems with traditional development methodologies with respect to deadlines and client satisfaction [37]. The core objective of XP is to succeed in reaching the client satisfaction (see more in *Appendix A. A quick overview of eXtreme Programming*). XP recommendations are oriented towards getting high quality software.

With XP, work is usually performed by small teams that include a customer continuously present on-site. As shown in Figure 3.2, XP development begins with the *planning game*, where customers and developers negotiate requirements in the form of *user stories*. The system concept is captured in terms of a *metaphor*, to support a single vision of success. The fundamental *short cycle time* (iteration) is no more than three weeks. Moreover, the technical process assumes *collective ownership* of the product so that anyone can work on anything. This fits with *simple design*, *pair programming*, and emergent design through *refactoring* to work. *Coding standards* are established by the team, and quality is assured through a *test-first approach* and *continuous integration*.

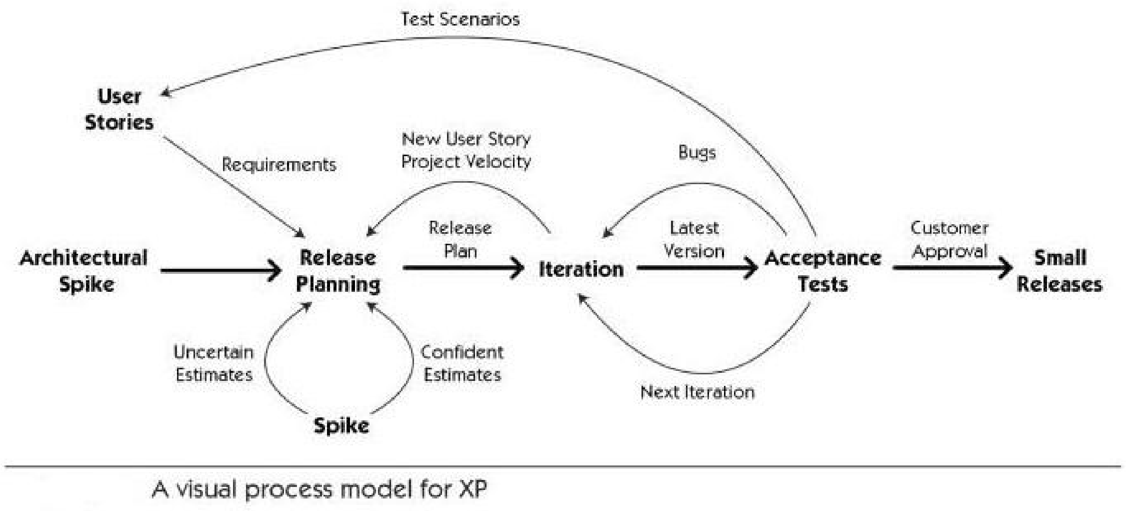

Figure 3.2. A visual process model for eXtreme Programming (adopted from http://www.extremeprogramming.org).

Table 3.3 presents common XP practices named differently by different authors ([2, 40]). The items in the lists are in order that compares the similar or same practices in the two literatures. As can be seen, the names of basic practices are mostly unchanged.

Beck K. [2]*	**Martin R.C. et al.** [40]
The planning game	The planning game, User stories
Small releases	Short cycles
Metaphor	Metaphor
Simple design	Simple design
Testing	Acceptance tests, Test-driven development
Pair programming	Pair programming
Collective ownership	Collective (code) ownership
Continuous integration	Continuous integration
40-hour weeks	Sustainable pace
On-site customer	Customer team member
Coding standards	Coding standards
Open workspace	Open workspace

Table 3.3. Different XP practices.

* Beck [58] adds one more practice that is *Just rules* (See Appendix A for more details).

Chapter 4: Literature background

The goals of this chapter is to discuss some main concepts related to the issues in the research questions (Section 2.4) as well as the main findings from literature study for answering those research questions (Sections 4.2 and 4.3). As a literature review should be complete and focuses on concepts [48], we first introduce the concepts by giving their definitions.

4.1. Definitions

Below, the following definitions of terms and phrases will serve to help readers understand the terminologies used in this research: customer value, customer satisfaction, software quality, cycle time, waste, CMM/CMMI, and ISO 9001.

4.1.1. Customer value

The common understanding of customer value is the customer's perceived trade-off between benefits (what the customer receives) versus sacrifices (what he or she gives up) while acquiring and using the product in a given situation [27, 31]. That is, customer value can be the level of return in the product benefits for a customer's payment in a purchase exchange, or in wider meaning, it is the judgment of value results from a trade-off between what the customer receives (e.g., quality, benefits, worth, ease of use,) and what he or she gives up (e.g., price, sacrifices).

Customers in this context are understood mainly as the clients who pay for the software (this also include the business-side people in each agile development team). So customer value means business value, which delivers profit to the organisation paying for the software in terms of, for example, the reduction of costs (less money), the improvement in service or performance (software quality), or the rise in revenue. However, the business cannot get value unless the software is used. That is the reason why the value creation process will also have to deal with the people who eventually use the software (end-users or target-users).

4.1.2. Customer satisfaction

Customer satisfaction generally refers to the degree of customer's positive (or negative) feelings about the value they receive as a result of using a product (or a service). In this research's context, customer satisfaction can be measured as the difference between the customer's (or client's) expectations on the output of the software contract and their experience. Therefore, *customer value drives customer satisfaction*.

4.1.3. Software quality

Software quality is commonly defined as the density of post-release defects in software program, which can be measured as the number of defects per thousand lines of code [18] [19]. The definition implies that product size (the number of lines of code) may play an important role in achieving high quality. Larger products with larger volume of code and increased software functionality provide more opportunity for the existence of errors. The possibility of defects is also increased by more modules and the degree of cohesion between them.

What is more, this definition is based on errors uncovered in system and acceptance testing, and these errors are determined according to customer specifications. There are certain levels of testing that are carried out to recognize software quality in software development life-cycle. First, unit testing (or component testing) refers to tests that conducted separately for a specific section of code (module or component) to ensure that specific function is working as expected. This white-box test aims to detecting and removing syntactical errors. Second, system testing tests the complete product to verify whether discrete modules when integrated together will work to meet requirements. Lastly, acceptance tests are carried out by real customers to make sure the product meets all their specifications.

In 1991, the International Organisation for Standardization (ISO) adopted ISO 9126 as the standard for evaluating software quality. ISO/IEC 9126, which deals with the quality assurance of the process used for developing software products, provides quality models (internal, external and quality in use [25]) presenting an approach to look at different quality attributes.

4.1.4. Cycle time

Typically, *cycle time* is the time to develop the product, i.e., the elapsed time in days from the start of design on the product until its final acceptance by the customer [20]. Cycle time includes certain software development life-cycle phases which are high-level design, detailed design, coding, unit testing, system testing and acceptance testing. According to Argawal and Chari [18], cycle time is important outcome variable because software projects are often carried out under strict delivery schedules.

In ASD, cycle time is iterative (which usually is called *iteration*). This is the cycle that starts with a list of requirements (or an idea) and ends with a finished working software (or a finished product).

In typical software development life-cycle, it may take months, whereas in ASD, it is usually a couple of weeks (2-4 weeks).

4.1.5. Waste

In broad understanding, any work and work product that is not directly contributing to the development of software should be considered as *waste* and thus should be avoided or minimized [12].

In this research, waste refers to obstacles get into the quick delivery path of software to customer, which we think includes waste in *cost, time, and effort*. It is obvious that the goal of any software team is removing waste from the development process, and that can mainly be done by changing the development process.

Development effort refers to the person-months (or days) required to develop the software product. It also includes cost and is often regarded as a surrogate for software development cost since personnel cost is the dominant cost in software development.

Figure 4.1 shows the relationships considered among waste, too much development effort, cost, and cycle time (see the definition of cycle time in **3.1.4. Cycle time**).

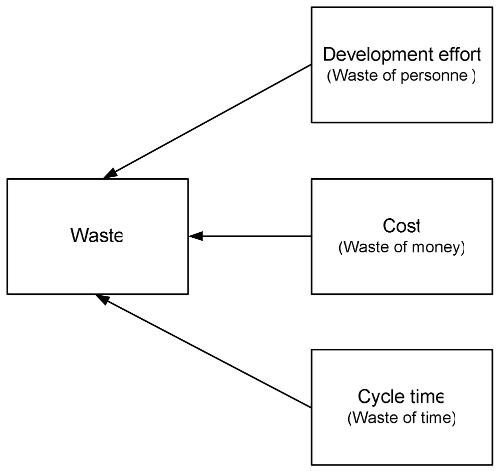

Figure 4.1. Waste decomposition. The arrows show the direction of contribution, e.g. waste in money contributes to the total waste of developing the software.

Figure 4.1 shows that in this research, waste consists of waste in development effort (personnel), waste in money (cost) and waste in development time. More waste of time and effort (more personnel) might lead to more waste of money (cost), but we do not examine those relationships. We only consider effort, money, and time as "ingredients" for estimating waste.

We summarize the definitions of the terms in Table 4.1 below.

Concept	**Definition**
Customer value	Customer's perceived trade-off between benefits versus sacrifices while acquiring and using a product in a given situation.
Customer satisfaction	The degree of customer's positive or negative feelings about the value he/she receives as a result of using a product/service.
Software quality	The density of post-release defects in software program.
Cycle time	The elapsed time from the start of design on the product until its final acceptance by the customer.
Waste	The obstacles get into the quick delivery path of software to

	customer.
Development effort	Person-months (or days) required to develop a software product.
Cost	Money spent on developing a software product (mostly on personnel)

Table 4.1. Definitions of the major concepts.

4.1.6. CMM-SW

The Capability Maturity Model for software (CMM-SW) was developed by Software Engineering Institute to describe the principles and practices underlying software process maturity. Its aim is to help organisations improve their software process maturity through an evolutionary path, from ad hoc, chaotic to mature, and disciplined. CMM-SW also helps assess how well defined the software development processes in an organisation are. A *well-defined process* is one that has readiness criteria, clear inputs and outputs, probably some standards, and procedures for performing the work (or separate phases). Moreover, there are also verification mechanisms as well as completion criteria (when it is completely “done”) for that process [21]. In CMM-SW model, organisations at level 3 possess defined processed.

The CMM is organized into 5 levels. Level 1 Initial is where the software process is characterized as ad hoc, or even chaotic in some cases. From level 2 to level 5, each level consists of a set of key process areas (KPA) that an organisation should focus on to improve its software process. Each key process area in turn comprises a set of key practices that indicate if the implementation and institutionalization (we will discuss more about the concept of *institutionalization* later in Section 4.3.3) of that area is effective, repeatable, or lasting (see Table 4.2). As a glance at Table 4.2 shows, the KPAs are clearly important to all types of software organisations. In addition, while the KPAs in CMM Level 2 are project-oriented, those from CMM Level 3 upwards are organisational-oriented.

CMMI-SW

Capability Maturity Model Integration Software (CMMI-SW) evolved from the multiple models of CMM as a single reference model to make it easier for organisations to follow the model requirements.

CMMI has two representations. The first one, staged CMMI resembles the previous CMM by discussing the five stages, and thus assist organisations who currently use CMM to migrate to CMMI. The second representation is continuous CMMI which is actually developed to bridge CMMI-SW to ISO/IEC TR 15504 (another software process improvement standard). Therefore, CMMI is still based on moving organisations from one level to the next one and gradually improving software processes.

CMM Level	KPAs
Level 5: Optimizing Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.	Defect prevention Technology change management Process change management
Level 4: Managed Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.	Quantitative process management Software quality management
Level 3: Defined The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organisation. All projects use an approved, tailor version of the organisation's standard software process for developing and maintaining software.	Organisation process focus Organisation process definition Training programme Integrated software management Software product engineering Intergroup coordination Peer reviews
Level 2: Repeatable Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.	Requirement management Software project planning Software project tracking and oversight Software subcontract management Software quality assurance Software configuration management
Level 1: Initial The software process is characterized as ad hoc, occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics.	[No key area]

Table 4.2. Key process areas in the CMM [21].

In this research, CMM and CMMI are mostly mentioned with their common characteristics. However, there are several parts in which we examine them differently, to the extent of their structure.

4.1.7. ISO 9001

ISO 9001 is the standard in the ISO 9000 series developed by the International Organisation for Standardization that pertains to software development and maintenance, identifies the minimal requirements for a quality system, when a contract between two parties requires the demonstration of supplier's capability to design and supply a product [35]. The two organisations could be an external client and supplier, or could be both internal like the finance (IT-demand side) and engineering groups (IT-supply side) in the same company.

Of the ISO 9000 series, ISO 9001 is the most directly relating to software development and maintenance. Organisations use it when they have to make sure that the supplier obeys specified requirements during several phases of development, including design (high-level and detailed), development, production, installation, and servicing. For applying ISO 9001 to the development, supply, and maintenance of software, guidelines are provided by ISO 9000-3. In fact, the standards are frequently used to register a third-party's quality system.

Requirements identified by ISO 9001 in terms of clauses. For example, in clause 4.2: Quality system, organisations are required to establish a documented quality system, including quality manual and plans, procedures, and instructions. Together with ISO 9000:3, this quality system is characterized as an integrated process throughout the life cycle.

As ISO 9001 implicitly addresses software improvement process at such minimum requirements, it would be quite flexible to apply practices and activities to meet its requirements. This leaves a space for the adoption of agile practices and methods. Since agile development is not an exact defined methodology, there exist a handful of agile methods variations that can be adopted in different organisations.

4.1.8. CMM versus ISO 9001

When it comes to comparing ISO 9001 and CMM, there are certain differences. First of all, ISO explicitly addresses customer satisfaction, while CMM assumes that adhering to the requirements of various key process areas implicitly leads to higher customer satisfaction (more in part **3.2.2. Maturity level and user satisfaction**). Second, CMM explicitly emphasizes on continuous process improvement, whereas ISO 9001 addresses only the minimum criteria for an acceptable quality system. Another difference is that CMM focuses strictly on software, while ISO 9001 covers much broader scope such as hardware, software, processed materials, and services.

On the other hand, there are also similarities between the two standards. The biggest similarity may be their bottom line: "Say what you do; do what you say". That is, they both require documentation. CMM-based processes must be documented and practiced as documented. It is the case that CMM KPAs are characterized by the phrases such as "conducted according to documented procedure" and "following a written organisational policy" [35]. ISO 9001, in turn, requires documentation containing instructions or guidance on what should be done (or sometimes how it should be done). Their similarities are further shown on a more detailed level, where some clauses in ISO 9001 are compliant to equivalent CMM practices. Every CMM KPA is related to ISO 9001 (at least weakly) in some way (see table 2). For instant, every CMM KPA at level 2 is strongly related to ISO 9001, and an ISO 9001 – compliant organisation would satisfy many CMM level 3 goals [35]. Since there exists the significant degree of overlap, an organisation should consider both while pursuing continuous software process improvement. Below (Table 4.3) we present the mappings found by Paulk [35].

ISO 9001 Clauses	**Strong relationship**	**Judgmental relationship**
4.1: Management responsibility	Commitment to perform	Ability to perform
		Verifying implementation
	Software project tracking and oversight	Software quality management
	Software quality assurance	
4.2: Quality system	Verifying implementation	Organisation process definition
	Software project planning	
	Software quality assurance	
	Software product engineering	
4.3: Contract review	Requirement management	Software subcontract management
	Software project planning	
4.4: Design control	Software project planning	Software quality management
	Software project tracking and oversight	
	Software configuration management	
	Software product engineering	
4.5: Document and data control	Software configuration management	

	Software product engineering	
4.6: Purchasing	Software subcontract management	
4.7: Control of customer-supplied product		Software subcontract management
4.8: Product identification and traceability	Software configuration management	
	Software product engineering	
4.9: Process control	Software project planning	Quantitative process management
	Software quality assurance	Technology change management
	Software product engineering	
4.10: Inspection and testing	Software product engineering	
	Peer reviews	
4.11: Control of inspection, measuring, and test equipment	Software product engineering	
4.12: Inspection and test status	Software configuration management	
	Software product engineering	
4.13: Control of non-conforming product	Software configuration management	
	Software product engineering	
4.14: Corrective and preventive action	Software quality assurance	Defect prevention
	Software configuration management	
4.15: Handling, storage, packaging, preservation, and delivery		Software configuration management
		Software product engineering
4.16: Control of quality	Software configuration	

records	management	
	Software product engineering	
	Peer reviews	
4.17: Internal quality audits	Verifying implementation	
	Software quality assurance	
4.18: Training	Ability to perform	
	Training programme	
4.19: Servicing		
4.20: Statistical techniques	Measurement and analysis	Organisation process definition
		Quantitative process management
		Software quality management

Table 4.3. Summary mapping between ISO 9001 and the CMM. Relationship columns show the CMM practices and features mapped to equivalent ISO 9001 clauses. [35]

Usage of CMM and ISO 9001

CMM and ISO 9001 are often used in critical software development life cycles (in the domain of *high cost of failure*) [68]. Their use is deemed appropriate in the cases of external strategic outsourcing of large projects, in-house large projects for low level transaction processing system such as integrated ERP solution, large contracts within defense (weaponry), aerospace (aircraft and spacecraft), and healthcare (safety-critical medical devices). This is because large projects are typically associated with significant risks (be it business risks or technical risks). For example, if a plane clashes, the cost of failure is extremely high.

4.2. Level of maturity and software value

In this section, we study from literature to find the answer to the following question:

What are the relationships between maturity level and software quality, waste reduction, value creation and user satisfaction?

Here, *software quality* is the focus of both SPI and agile methods, but the concepts of *waste reduction*, *value creation,* and *user satisfaction* are of the main benefits that agile methods promise.

The next sections summarize the review of previously published work on the relationships between maturity level and those factors.

Conventional wisdom suggests that there are conflicting influences on software development effort, quality, and cycle time: cycle time may be compressed at the cost of quality, experienced professionals may improve quality but at increased costs, quality may be achieved at the cost of increasing testing effort, larger team size may reduce development time while raising total costs, process maturity may improve quality but at high cost... However, the review from literature reveals an alternative perspective of getting "both".

4.2.1. Maturity level and value creation

As mentioned above, the general definition of customer value is the customer's perceived trade-off between benefits (what the customer receives) versus sacrifices (what he or she gives up) while acquiring and using the product in a given situation [31]. Therefore, the essence of value creation is that value emerges in customers' working processes, when they make use of the solutions they purchased. In other words, value is not created in the development and production process, but determined by customers. Furthermore, customers perceive value differently in different contexts, and the product developers support customers create their own values.

It is obvious that people choose a product according to the value that the product brings to them in use. Johnson and Weinstein [32] also add "a strong competitive advantage can be gained through consistently providing superior customer value" (pg.10). In the software industry, development organisations (teams or companies) do recognize the importance of customer value and try to enable value creation. Value-based approach for requirements engineering has been widely used and further applied in value-based release planning context while studying stakeholders' perspectives and values influencing requirement selection [22, 27, 29].

What is more, since it is generally accepted that product features form the core of value creation, an implicit assumption is that more value creation is perceived while companies deliver more product features (and faster) to customers (by doing so they also bring better customer satisfaction). In the next sections, the discussion is about the increase in product quality, as well as the decrease of development cycle time, and cost due to higher maturity level. From the standpoint of the author of this research, value creation for the customers also means that the development organisations meet customer's quality, delivery and cost expectation. Therefore, those viewpoints all suggest that higher maturity indirectly increases value creation.

It could be also argued that quality models (e.g., CMM/CMMI, ISO 9001, ISO 2000, SPICE/ISO 1554, etc) are concerned with the "means" and put little emphasis on the need to understand "value", while agile thinking is focused on "value" and the results (i.e., the "ends"). Agile methods (such as Scrum) do define and prioritize the working products to be delivered incrementally to customer, with the highest value first. Therefore, it is possible to make a conclusion that high maturity processes still need agile practices in order to best deliver customer value.

4.2.2. Maturity level and customer satisfaction

This research mainly focuses on the concept of customer satisfaction, which is strongly related to the concept of customer value. According to the understandings of customer satisfaction and customer value in this context (see more in 3.1. Definitions), customer satisfaction tells the development organisations how much more they have to do, while customer value can show more concisely the direction on what to do.

The strong relationship between customer value and satisfaction is illustrated in both some certain facts and some beliefs. First of all, in many cases, the more cost and time the development organisations save for clients, the more value the clients perceived to be delivered or the more satisfaction is achieved. This again lets us think that, although customer value and customer satisfaction are subjective and not defined by the software suppliers, the software production process itself still has significant impacts on the customers' perceptions of value being delivered and the level of satisfaction with the product. Secondly, there exists a held belief that the more satisfied the customers are, and the more value is added for their benefits, the greater their loyalty, the more secure their retention and the better the supplier's image and reputation [34]. In this way, customer value and satisfaction both contribute to the development organisations' competitive advantage.

As mentioned above, expectations play a significant role in customer satisfaction. Jones et al. [33] state that "when expectations are met or exceeded, customers report higher level of satisfaction" (pg. 11). That could explain why expectation management has always been stressed in software development projects.

One of the basic facts from the business realities in the software industry is that everything in software engineering boils down to user satisfaction, and this satisfaction is conditional on the overall behavior of the system, with software is in the first place. In this way, user satisfaction is part of software quality in use.

Generally, a software development team would ensure customer satisfaction by providing predictable, high-quality software products that are delivered on schedule [23]. When companies grow in terms of maturity, cases from previously published studies indicate significant increases in customer satisfaction. One example is the case of Telcordia [23]. This study reports that customer satisfaction rises from 60% (in 1992, when the company was not certificated at all) to 95% (in 1997, achieving ISO 9001 and being assessed at CMM Level 3). In this case, Telcordia's customer satisfaction is deemed "complete" by clients being provided with predictable, high quality software products that are delivered on schedule. Another example is the result of the survey by Herbsleb et al. [16], in which ratings of customer satisfaction reach 100% at CMM level 3 (Defined), from 80% at the level 1 (Initial).

4.2.3. Maturity level and software quality

A number of approaches have been proposed to improve software quality. These include TQM, Six Sigma, and CMM. The basic idea behind all these approaches is to identify ways to improve quality in a given situation.

Much literature shows clearly the focus of SPI standards on quality assurance. Firstly, in a published study [18], Agrawal and Chari emphasize the idea that SEI-CMM has been one of the most popular efforts in enhancing software quality and reducing development costs. Secondly, in the CMM model, we observe that software quality is first positioned in level 2 with software quality assurance, and then in level 4 with software quality management for product and process quality.

Previously published studies [16. 17, 19, 20] also provide strong evidence confirming the significant impact of higher maturity on software quality. For example, Herbsleb et al. [16], find that organisations engaged in CMM-based SPI tended to improve substantially in quality, cycle time, and productivity. Their empirical study shows that the performance of product quality also improves with increased maturity levels, and reaches 100% of excellence at level 3. In another study, Krishnan [17] showed that process maturity significantly increased quality (but did not show evidence of direct effect on cost).

There is also quantitative empirical evidence that supports the fact that improvements in process maturity lead to higher quality [20]. There, Harter et al find that a one percent improvement in process maturity was associated with a 1.589 percent increase in product maturity. Furthermore, Diaz and Sligo [19] explain also the relationship between software quality and maturity levels and report that in the case of Motorola, each level of the CMM improves quality by a factor of about 2. The improvement in quality is expected for projects that transition from level 2 to level 3 due to the KPA Peer Review in level 3. Peer Review has been widely recognized as one of the most important factors in detecting and preventing defects in software products. From level 3 to level 4, quality is improved through the KPAs Quantitative Process Management and Software quality management. As a consequence, when a software unit at Motorola has upgraded from CMM level 2 to level 5, the average defect density reduced from 890 defects to 126 defects (per million assembly-equivalent lines of code).

4.2.4. Maturity level and cycle time

Since examining literature does not define directly the concept of waste reduction, this research conceptualizes it from three components: effort, cycle time, and cost. This section starts with cycle time.

The authors of existing research [16, 17, 18, 19, 20] believe that short cycle time is the major measure that can truly make development teams different, and reducing cycle time means getting a software feature into the hands of a customer as fast as possible.

There are two different perspectives of viewing the relationship between process maturity, cycle time, and software quality [20]. One view (we refer to it as the ‘conventional way’) is that cycle time must be traded off against improvements in quality. That is, while trying to increase higher process maturity, more time and effort required for testing, code inspections (ensuring quality), and for creating documentation. However, there has been also an alternative perspective that improvements in quality can lead to overall reductions in cycle time and development effort [17, 18, 20]. This is because in the context of software production, several studies have indicated that more time is needed to fix defects uncovered at later stages in software development than in earlier stages, but adopting disciplined practices (like described in CMM family) means focusing on preventing and uncovering defects early in the development process, so that less time wasted or spent in later stages.

Findings from the examined literature show the positive and direct effect of process maturity on cycle time, but product quality is significantly and negatively associated with cycle time. Interestingly, the net effect of process maturity on cycle time is negative. Based on the measurements at Motorola, Diaz and Sligo [19] report that the cycle time is about eight times faster at CMM level 5 than at CMM level 1. The similar time savings are also reported from process improvements by Harter et al. [20].

4.2.5. Aggregation of the findings

We present the results of the findings in a model that aggregates them and highlights how the aspects that we studied relate to each other. The outcome of this analytical exercise is depicted in Figure 4.2. In the figure, the arrows show the direction of direct effects, and the signs “+” and “-” tell the effect is positive or negative, respectively. For example, higher process maturity leads to higher value creation, cycle time, and customer satisfaction; or higher software quality (gained by higher process maturity) leads to less cycle time and less development effort.

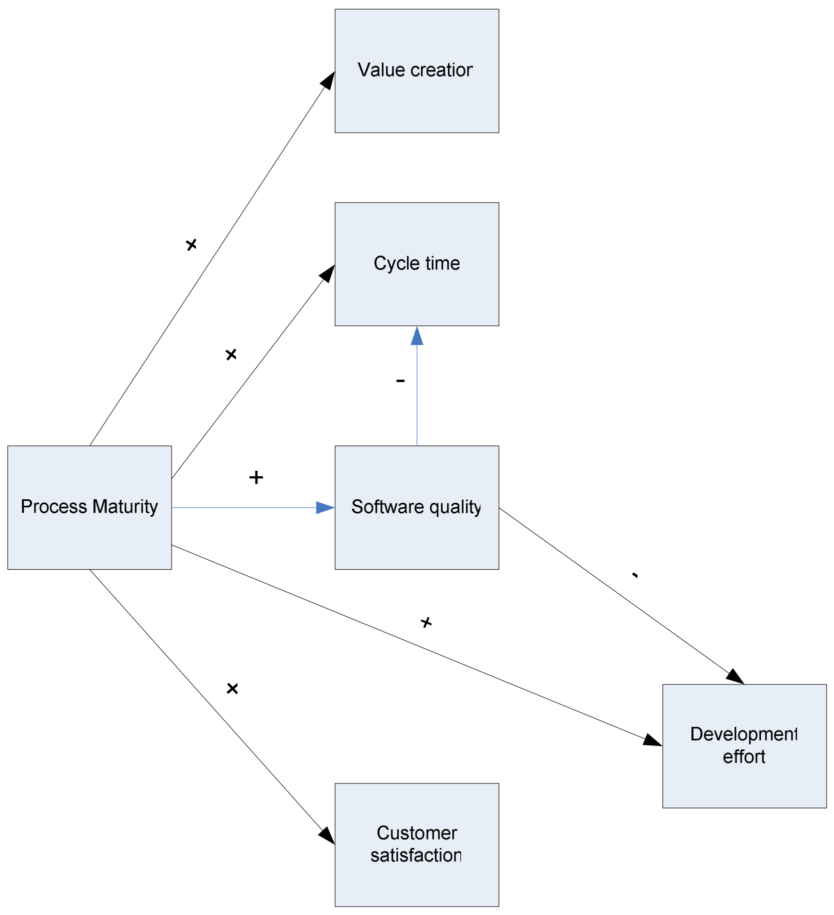

Figure 4.2. Research results question 1 in details.

4.2.6. Maturity level and waste reduction

By its definition (section 3.1.Definitions), waste reduction deals with reducing waste in *development effort*, *cost* and *cycle time*.

It is obvious in software engineering that *reusing software components* can reduce the time and the costs of developing a software application (and can also improve the quality of the product because the existing components are often tested well already). Nevertheless, in this part the more general context is looked at, where the development can be carried out from scratch.

4.2.6.1. Process Improvement reduces cost

Krishnan [17] finds that higher product quality significantly reduced both development and support cost, so to the extent that process maturity increases quality, process maturity may also indirectly decrease cost.

4.2.6.2. Process Improvement reduces effort

Several previously published studies report that higher process maturity can reduce development effort. First, Harter et al. [20] point out that, similarly to the findings in the cycle time model, higher levels of process maturity (as assessed by the SEI's CMM) are associated with significantly higher product quality, but also with increases in development effort. However, the marginal reduction in effort resulting from improved quality outweighs the marginal increase from achieving a higher level of process maturity. Thus, *the net marginal effect of process maturity is reduced development effort.* Presumably this is because of the reduced rework and the improved understanding of software requirements. Second, in another specific research on quantifying the effects of process improvement on effort, Clark [36] finds that one level change in process maturity reduced effort by 3%-15%. Some examples of the effects are, according to a SEI report [37], Boeing Australia achieved a 60% reduction in work, while Lockheed Martin had a 30% increase in software productivity.

Together with the result that the net effect of higher process maturity on cycle time is negative, we depict the results in Figure 4.3.

Removing or reducing waste from the development process also fits with the philosophy of getting the right thing the first time, or at least detecting and fixing defects as early as possible. The later stages that defects are uncovered, the larger waste of time and effort exists. This suggests two tasks need to be done. The first one is that testing needs to be carried out very early in the development process. The second task is to fully analyse the root cause of a non-trivial defect whenever it is found, to make sure that the same kind of issues would never occur again.

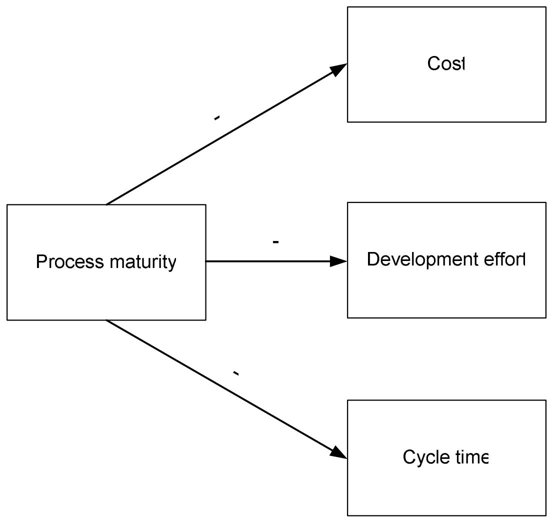

Figure 4.3. The relationships between process maturity and cost, development effort, cycle time. Higher maturity of software development process leads to lower cost, less development effort and shorter development cycle time (they are indirectly effects through higher software quality gained due to the higher process maturity).

4.2.7. Adding the new findings to the aggregation model

Integrating Figure 4.1, Figure 2, and Figure 4.3 yields the relationships depicted in Figure 4.4. The figure shows that the higher process maturity of a company, the higher value the company creates to customers and the less waste exists. Moreover, by improving the software development process, the company develops higher quality software and gains more customer satisfaction.

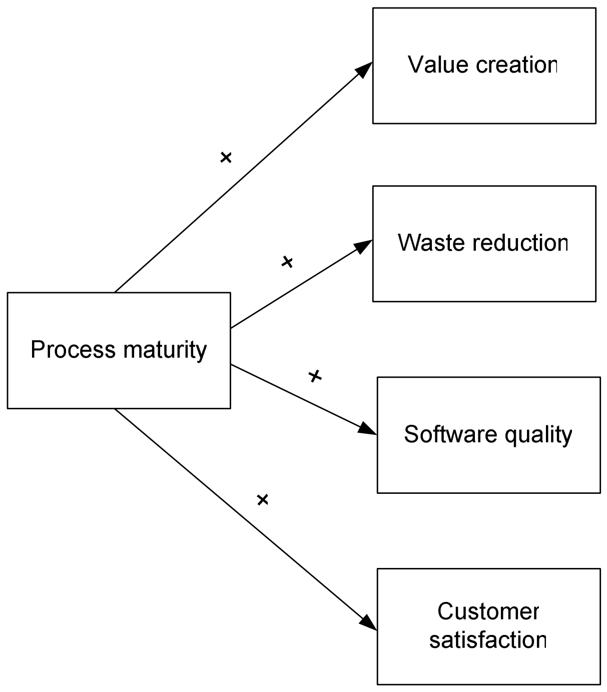

Figure 4.4. Model for research question 1 in abstract level.

4.2.8. Conclusions and discussions

Based on the findings presented earlier, we think that the following conclusions are justified to draw:

First, it is interesting that higher process maturity has a positive effect on waste reduction, which is contradictory to conventional thought. At first, in order to get more mature, organisations have to spend more time, effort and money on implementing some rigorous methods, and on documentation. However, when their software development process is already more mature, the software quality gets much better, and there is less rework required in the later stages of the process, and the overall time and effort are reduced. This is the strategy of "investing now to gain much more in the future".

Second, higher level of maturity achieved by proper and wise implementation and adaptation of the SPI standards can increase software quality, user value, user satisfaction, as well as decreasing waste. However, the achieved results are context-based. For example, in [20] (which has the same results) the context is customer software development of an algorithmically intense system in a COBOL, mainframe development environment.

Third, as both customer value creation and satisfaction are customer-dependent, defining these concepts is not straightforward and, in fact, quite complicated. However higher maturity tends to be significantly positive determinant of these two factors. After all, most development organisations pursue higher maturity in order to get more competitive advantage and to deliver the best possible systems for their customers' budget of time and money.

Fourth, one of the sub-results is that development effort and quality are affected by project size alone. It is the case that larger projects are likely to have more defects. So software size is the most significant factor that affects development effort, quality, (and cycle time). Moreover, there are some other factors also involved when examining these four factors, such as *requirements ambiguity*, *product size*, *product complexity*, *team size*, *schedule pressure*, and *personnel capability*. These factors will help us better connect issues of high maturity level with those of agile methods. We believe that under schedule pressure, developers are likely to trade-off quality, and personnel quality of capability would be also a significant predictor of software quality.

4.3. Level of maturity and Agility

The goal of this section is to find the answer to the RQ2 from previously published studies. The RQ2 is:

What is the relationship between process maturity models and ASD methods?

To help answering this question, we formulate three sub-questions:

i) Does process maturity models fit with agile methods?

ii) How can agile methods benefit from higher level of maturity of an organisation?

iii) How can agile methods be adapted in organisations that have high level of maturity?

With these questions, first we would like to examine whether an organisation can achieve high maturity level (based on CMM or ISO 9001, for example) using agile methods, or whether the SPI standards contradict with agile methods in some way. In other words, we study the interrelations between agile methods and process maturity models to see if SPI standards and agile methods can co-exist in an organisation, so that the organisation can take advantage of both of the approaches (we already examined the effect of having higher maturity level on some aspects of the software organisation by answering RQ1). Moreover, we investigate how agile methods are adopted if the organisation achieves higher maturity level, as well as what the agile methods need to adjust in order to adapt with that level of maturity.

Since there are a lot of similarities between ISO 9001 and CMM (section 4.1.8), and the majority of previously published studies has taken CMM to examine the relationships with agile development (we think that this is because CMM is the most rigorous quality standard world-wide for software, with the most accurate and defined areas for applications), in this section we mainly take CMM as the representative for process maturity models. However, there are also other studies about the relationships between other SPI standards and agile methods, which will also be mentioned. One example is the study of ISO 9001 and Agile development [8] (Scrum and XP), in which McMicheal and Lombardi conclude that ISO 9001 is not only compatible with agile, but can provide just enough structure to help ensure agile processes are followed. Another example is the study with SPICE or ISO/IEC 15504 [7]. The study shows that in principle, ISO/IEC 15504 can be effectively used also in agile contexts if genuinely applied.

4.3.1. The CMM – Agile fit

As mentioned earlier in the introduction (Chapter 1), SPI-based standards and agile methods are different. The former are frameworks for process improvement, while the later is a development philosophy. For example with CMM, agile fits within the constructs of CMM. CMM tells what needs to be done, but does not say how to do it. An ASD method provides a set of best practices to guide you how to do. Within each level of CMM, there are goals and practices. For each of these there are approaches that can be used and that are consistent with agile methods and there are approaches that are not.

To give an example, we can take a look at the first goal *Obtaining an Understanding of Requirements* in the very first practice in CMM Level 2 – *Requirements Management*. Traditionally, people develop a comprehensive document of SRS (Software Requirement Specification) which consists of all requirements that come to agreement with customers before starting the design. However, in many agile methodologies, people write customer's stories and have them approved one at a time. Each iteration in agile methodologies consists of write-approve-develop customer's stories. Both approaches (i.e., using SRS and using user stories) are valid and consistent with CMM framework. Agile approaches are even simpler in execution (the *simplicity* principle).

In the following sections we investigate in more detail how the CMM framework fits with agile methods, in particular, with the two most popular methods XP and Scrum.

4.3.1.1. XP and CMM

The CMM-SW focuses on both management issues involved in implementing effective and efficient processes and on systematic process improvement. XP, on the other hand, is a set of practices – a methodology – that is effective in the context of small, collocated teams with rapidly changing requirements.

As XP is more a development approach (focuses on software engineering process) than a management one, it is more suitable to technical work than to management and infrastructure issues and more suitable to premature software organisations. Therefore, XP practices are addressed mostly in CMM level 2 and 3, as shown in Table 4.4. (See Appendix A for more explanation of these XP practices)

CMM Level	**KPAs**	**XP Practices applied**
Level 5: Optimizing	Defect prevention	Continuous integration
	Technology change management	Not applied
	Process change management	Not applied
Level 4: Managed	Quantitative process management	Not applied
	Software quality management	Not applied
Level 3: Defined	Organisation process focus	Just rules
	Organisation process definition	Metaphor
	Training programme	Not applied
	Integrated software management	Not applied
	Software product engineering	Metaphor, Simple design, Testing (unit and functional), Refactoring, Coding standards
	Intergroup coordination	On-site customer, Pair programming
	Peer reviews	Pair programming
Level 2: Repeatable	Requirements management	On-site customer (requirements change, user stories) Continuous integration (iterations, system and acceptance testing)
	Software project planning	Planning game, Small releases
	Software project tracking and oversight	Small releases ("big visual chart"/burn charts, project velocity, commitments/ stories)
	Software subcontract	Not applied

	management	
	Software quality assurance	Pair programming (pair pressure)
	Software configuration management	Collective ownership, small releases, continuous integration, testing

Table 4.4. XP practices and CMM KPAs (adapted from [9, 37, 51, 55]).

At the level 2, *subcontract management* is usually not applicable to small organisations (XP's target environment), but only to organisations that do subcontracting. (For other applicable areas, Scrum can also contribute together with XP to fulfill those process areas, except for Requirement management [37]). Relates to the process area *Software project tracking and oversight*, XP emphasizes on 40-hour weeks which is general human factors concern, although CMM does not address it, having "rational work hours" is usually considered as the best practice. Though XP practices meet many requirements for CMM level 2, it does not fulfill CMM requirements yet. However, it is possible to construct a process that fulfills CMM requirements by adding practices to XP [54].

At level 3 of CMM-based context, since organisational assets are outside the scope of the XP method [9], XP cannot address *integrated software management* (there may not be any organisational assets to tailor). For *software product engineering*, even though there are several XP practices that address it, the design documentation is a concern in many contexts (e.g., hard real-time systems, largely critical systems, or virtual teams).

One important finding of Alegría and Bastarrica [37] is that it is possible to reach maturity level 2 using agile methods, or in other words, it is possible for organisations to achieve a CMMI certification implementing agile methods (in this case the agile methods are XP and Scrum, or XP@Scrum [37]). Vriens [53] also reports the case of the department Software Engineering Services (part of the Philips Research organisation in the Netherlands) which got certified for both ISO 9001:2000 and CMM Level 2. However, since some process areas, mainly those of the maturity levels 4 and 5, are in conflict with agile principles [51], agile methods can be applied without any major adaptations up to level 2 and up to level 3 with some minor changes. This does not mean that XP@Scrum is incompatible with higher level of maturity (level 4 and 5), because if a company achieves CMM level 4, it has been already certified for level 3 (and lower levels), and as Alegría and Bastarrica also mention the case of Motorola Argentina, a company certified as CMM level 5 and applied XP@Scrum successfully to make its development process more agile.

In another study, Paulk [9] evaluates XP from a CMM perspective, and concludes that XP includes very good engineering practices. He also states that taken or combined together, the two methodologies (i.e., XP and CMM) can generate synergy, particularly in conjunction with other good engineering and management practices. While XP provides a system programming perspective, CMM provides an organisational process improvement perspective. The idea is that companies can take advantage of both of them by adapting and adopting their practices. Moreover, the study also implies that Scrum can be a good candidate as the supplement for organisational and managerial issues and practices.

4.3.1.2. Scrum and CMM

It is shown that while working with agile methods, CMM Level 2 and 3 can be achieved as well (with some extra effort). This holds for XP and also does for Scrum, especially at CMM level 2 which overlaps with agile way of working. The overlap dues to the fact that both focus on basic project management. In this level, the focus on people in Agile and the organisational focus of CMM are nicely complementary.

As stated above, Scrum deals more with project management issues. That is why in this section we map Scrum practices with CMM project management process areas (Table 4.5).

Level	**Process areas**	**Agile practices**
Level 4: Managed	Quantitative Project Management	Not applied
Level 3: Defined	Integrated Project Management	Not applied. Scrum does not define a set of organisational processes.
	Risk Management	Informal manner because there is not strategy for risk response based on historical data
Level 2: Repeatable	Project Planning	Work breakdown structure (WBS) Project Backlog (pre-defined sprints) Sprint backlog
	Project Monitoring and Control	Burn-down charts Scrum meetings
	Supplier Agreement Management*	Not applied
	Software quality assurance	Scrum retrospective

Table 4.5. Scrum practices and CMM project management process areas (adapted from [39, 55]).

** For CMMI: instead of Software Subcontract Management in CMM, CMMI has the KPA Supplier Agreement Management.*

From CMM perspective, Scrum is just one process out of a set of processes used to execute a project. In a CMM context all processes for development are monitored for effectiveness and efficiency.

Jakobsen and Sutherland [43] conclude that using CMMI and Scrum together results in significantly improved performance while maintaining compliance with CMMI. In their case study, Scrum reduces every category of work (defects, rework, total work required, and process overhead) by almost 50%.

In summary, we are convinced that CMMI (and also ISO 9001) is an appropriate way to improve processes in an agile environment. However, the best improvement approach in an agile environment is to stop at CMMI level 3, since there are several process areas (most of level 4 and level 5) which are in conflict with agile principles. If we tried to carry out the improvement up to level 4 or 5, we would weaken the agile method and eliminate several of its benefits. Moreover, such actions would be contradictory to the aim of CMMI as well, i.e. improving the process by making the agile method as good as possible and not turning it into a different kind of method which is not agile anymore.

4.3.2. Mutual benefits of combining high level of maturity and Agile

As many published studies have stated that a CMM-SW certification benefits many type of companies [13, 16, 19, 20, 21, 23, 51, 55, 62], it is intuitive to conclude that it benefits agile-based organisations as well.

First, a process maturity model, like CMMI, ensures that all relevant and required processes are considered or addressed in an agile environment [13]. Agile methods work best in a disciplined culture, or in other words, the value from agile methods can only be obtained through discipline use [13, 37, 53]. The two most popular agile methods – XP and Scrum – are highly disciplined methodologies, emphasizing verbal over written communication [41, 53], encouraging frequent inspection and adaptation, teamwork, self-organisation and accountability. That is, they are disciplined by their principles and practices, e.g. continuous integration, on-site customer, immediate feedback etc. CMMI, in turn, ensures that all relevant processes are considered to maintain a disciplined mature organisation. An agile culture with a disciplined approach can systematically improve agile velocity (the amount of work that a team can complete in a specified period of time) and quality using proven CMMI techniques [43], especially with the presence of skilled people and good leadership.

Second, projects combining agile methods with SPI standards combine adaptability and predictability to better serve large customer needs [13, 43]. Agile approaches themselves can actually considerably improve predictability while keeping their schedule commitments because schedule variation (if exists) is contained within each iteration [68]. However, CMM provides insight into what processes are needed to maintain a disciplined mature organisation capable of predicting and improving performance of the organisation and of its projects. Moreover, APM methods like Scrum also provide guidance for efficient management of projects in a way that allows for high flexibility and adaptability.

Third, the CMM's approach to organisational *institutionalization* (i.e., establishing the culture that "this is the way we do things around here" [56]) can help to scale up agile practices by making use of plans and considering process documentation. Besides being a framework, CMM is also a collection of process area components that are institutionalized across an organisation [73]. The concept of institutionalization, which is crucial for CMM-SW, is the element that is missing from agile methodologies. Agile is seen as wonderful for small organisations and small (10 to 15 people), separable projects, but its ability to scale up to large projects and organisations can be debated. When a project has more than 20-40 people, some kind of scale-up strategies must be applied.

Fourth, we believe that process maturity models help agile teams better transfer knowledge. Adopting agile practices alone, organisations tend to focus on verbal communication between the project members and between co-working organisations. In that situation, tacit knowledge (see more in section 3.2.2) is commonly used to build process knowledge. With the base of the CMM, certain level of documentation and measurement (explicit knowledge) is required and also the efficiency of communication is likely to be more ensured [56]. Requirements document (or Product backlog, user stories in case of agile practices) or technical specification will exist and be available for developers and testers, as well as for newcomers to get acquainted with that kind of project and its processes. Nonaka [74] describes four basic patterns for creating and sharing knowledge in any organisation: 1) from tacit to tacit: when applied to our context, it is what Agile focuses on; 2) from explicit to explicit, which is the focus of CMM; 3) from explicit to tacit; and 4) from tacit to explicit. Therefore, being both Agile and CMM-based would help the organisation better transfer knowledge among individuals and from individuals to documented processes and practices.

Fifth, a SPI standard can help the project team better gain trust among all the members, including the customer. The team then can count on their process maturity (which is certified) to provide confidence in their work [50]. In principle, the team itself can not work effectively and efficiently without a high level of trust [1].

On the other side, agile methodologies can also help SPI-based organisations improve customer satisfaction, get higher quality for their products due to the focus on value and immediate feedback from customers. That is, Agile help a highly mature organisation improve value creation by better focusing on customers and their real desired values. Jarvis and Gristock [5], Sutherland et al. [13] quantitatively present evolution of indicators of quality and productivity with the adoption of agile merged with CMMI. Gravis and Jarvistock [5] state that (1) defects reduced by 66%, critical defects reduced by 79%; and (2) duration (in days) reduced by 44% and effort (in hours) reduced by 47%. Sutherland et al. [13] report that Scrum reduced defects by almost 50% compared to their previous CMMI level 5 implementation and projects using Scrum shows a 201% increase in productivity. In another research, Alleman [73] also states that the search for better development methods is the goal of CMMI and agile methods bring faster and cheaper attributes to this search.

4.3.3. Implementation of agile methods into mature organisations

Section 4.3.1 shows that agile methods can be really adopted while maintaining CMMI compliance. CMMI is always a framework for improving the processes of software organisations [73]. The framework defines the goals that agile practices can be employed to achieve the goals.

As the value from agile methods can only be achieved through a disciplined use, the introduction of ASD and/or AMP methods in a highly mature organisation is made easier by the matter that discipline is “institutionalized”. By this, we mean that the CMM family has a concept of *institutionalization* that can effectively help establish discipline needed to make use of agile methods. Institutionalization is defined in CMMI as “the ingrained way of doing business that an organisation follows routinely as part of its corporate culture” [13]. Others have described institutionalization simply as “this is the way we do things around here”. It is noteworthy that institutionalization is organisation-level concept that supports multiple projects.

CMMI supports institutionalization through the generic practices associated with all process areas. As part 4.3.1 talks about the ability to have agile practices in CMM level 2 and 3, we can also look at generic practices associated with these levels in CMMI and how they might help organisations use agile methods. Sutherland et al. [13] list 12 generic practices associated with maturity levels 2 and 3 in the CMMI to show that the practices have at least partial support in Scrum or other agile methods.

To illustrate, Sutherland et al. [13] provide an analysis of the effect of introducing Scrum practices into Systematic – a CMMI level 5 independent software systems company with offices in Denmark, USA and the UK. The company initiated four pilot projects which resulted in the decision of adopting Scrum and story-based early testing. These methods are now the default choice for new projects, and are integrated in the process descriptions at Systematic. An important insight for Systematic was that adoption of these agile methods involved only small adjustments to existing processes. Scrum was institutionalized at Systematic over a period of approximately six months [43]. The combination of Scrum and CMMI level 5 doubled productivity and cut defects by 40% compared to their previous CMMI level 5.

4.3.4. The adaptation of agile methods in organisations that have higher maturity level

One of agile methods' virtues is that we can change and improve them for different environment [9, 12, 42, 45]. There exist a handful of agile methods that varies, and each method can also be applied flexibly in each specific context, as long as it is still based on common principles described in chapter 3. While getting higher maturity and implementing some SPI standard, organisations that follow agile strategy should take advantage of the good ideas in both agile and the SPI standard, and exercise common sense in selecting and implementing those ideas. The two approaches also need to adjust to adapt to each other, and this has more to do with agile methods due to their flexibility, and because the SPI standard is more about *what*, whereas agile methodology is more about *how*.

The adaptation of an agile process can happen right at the start of the development project which covers 1) the requirements and objectives for quality, and 2) the need to establish processes, documents and provide resources specific to the product (which are normally required by standards of capability) [12]. This adaptation is about process planning, which may include deciding iteration length, strategies for requirements documentation (e.g., formal or not formal, rigorous or simple), effort or staffing etc. This may also include the number of agile practices used and the degree of commitment to those practices. For example, if the degree of requirements' formality is high, then the team may less use Srum boards. The planning of an agile project (about defining how the methodology will be used in the given project) can be easily documented in a simple form specifying, for instance, iteration length, how to record and track requirements etc.

On the other side, it is also intuitive to assume that the SPI standard should also adjust to adapt with the agile environment, when it works in an agile environment.

First, SPI-based processes would become less document-centered [12, 55]. It is not necessary for an organisation to have heavy procedures or a lot of documentation to show they are working on a certain level of CMM (e.g. Level 2) or be ISO 9001 complient. As stated in [21]: you need to look at what people are actually doing. In most organisations the work practices are different from the work procedures. Therefore, the combination of the best engineering practices and good enough necessary documentation would be ideal [55]. Also, to be CMM-Agile, it is critical for an organisation to balance stakeholders' involvement, documentation, interaction and communication in the project.

Second, SPI-based processes would encourage more collaboration or tacit knowledge sharing in order not to destroy the very concept of agility. The central role of tacit knowledge in agile method has been highlighted [3, 54]. If a project team is organized into single or multiple small and collaborative teams (for example of 5-7 people as adapting Scrum), developers will be able to share tacit knowledge about development processes (including, for instance, coding skills which can be shared through pair programming in XP). In that way, the organisation will provide really an excellent training environment for all parties.

To sum up, there is no need to change neither SPI standards nor agile concept. We do not need to "re-invent the wheel" while lots of good practices are already proven. We only need to flexibly and wisely apply them to achieve the conformance. That is, there is a need of taking advantages of existing best practices in both approaches while adopting each other.

4.3.5. Conclusions and discussion

Conventional thinking would conclude that agile and SPI standards or models like ISO, CMM, and SPICE must not be compatible, since the standards are often characterized as being heavy on process or heavy on documentation – the opposite of agile. The conventional assumption that agile is about no documentation is not actually true, as well as the assumptions that the standards need to be a burdensome process. The findings presented earlier show that the standards are not only compatible with agile, but can also provide a framework in which agile processes and practices can be employed.

The findings also show that it is possible to achieve CMMI levels 2 and 3 or ISO 9001 certification with agile methods and some level of their adjustments plus some traditional elements (plan-driven processes and practices). *However, there are certain challenges for an agile environment to overcome in order to be CMMI compliant*. First of all, both ISO 9001 and CMM require the existence of a (independent) quality assurance group to provide management with appropriate visibility into the process being used [53]. This requirement is not easily met since Agile encourages team self-organizing, and peer pressure from pair programming can be strongly effective to achieve a high level of quality but it does not necessarily give management full visibility into all issues. Second, there is a difficulty to carry out assessments of projects which use agile methods, since the specific practices suggested by CMMI often differ from agile approaches [51]. For example, there are interpretation issues concerning knowledge management. One question the auditor has to answer is whether an alternative practice relying on tacit knowledge, and can be considered working and institutionalized, is acceptable for CMMI or not? [54]. To remedy this situation, a catalogue of practices that are typically used by agile methods to implement CMMI goals, as well as common guidelines for interpreting adequacy of agile practices should be developed. Until then, the auditors may end up with different assessments.

The existing published studies have the limitation that they show only the feasibility of achieving CMM level 2 and 3 by using agile methodologies (although at low level - agile practices). There are some KPAs at the higher level (than level 3) which agile practices and be applicable to, but not all and hence not enough to get to higher level. As summarized above, only one KPA in level 5 (Defect Prevention in CMM or Ensure continuous improvement to support business goals in CMMI) is supportive to agile methods.

The benefits coming out of the synergic combination of Agile and SPI standards are only the internal motivation of an organisation that is originally agile and wants to achieve more maturity, to become more efficient and effective. Another driver – that we can call an external motivation - that we think is of no less importance is from customer inquiries. Many customers of software companies require appropriate levels of documentation around the companies' processes to demonstrate and prove their capability. Having enough motivation would help the organisations really become more mature and reduces its cost, increases its profits as well as delivering more value to customers.

Chapter 5: The model

Based on the findings presented in Chapter 4, we come up with a framework as depicted in Figure 5.1.

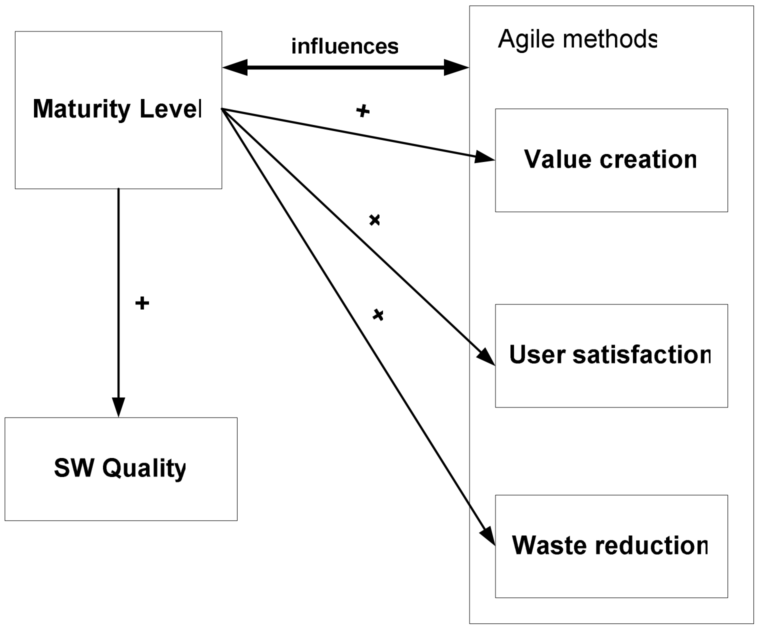

Figure 5.1. The research framework.

Figure 5.1 shows value creation, user satisfaction and waste reduction are advantages of agile methods, and maturity level has positive effects on them. That is, the higher process maturity, the higher quality software delivered, the higher user satisfaction gained, the less waste exists, and more value created to customers/clients.

The “influences” relationship between maturity level and agile methods represents that (Section 4.3):

- Agile methods and process maturity model are compatible,
- Process maturity model can be achieved using agile methods,
- Agile methods and maturity level can benefit each other,
- and they can adjust to adapt to each other.

Chapter 6: Case study

6.1. Introduction

After the literature study some open questions arose, and we formulated them as research questions (RQ):

RQ1: Which software development and management processes do organisations have in order to be compliant with their current level of maturity?
RQ2: Which agile processes or practices do they have?
RQ3: How processes in RQ1 and RQ2 co-exist or impact each other?

In order to answer these research questions, we conduct a case study. The definition of a case study as a research strategy can be developed in two parts, according to Yin [14] (p. 13):

1. A case study is an empirical inquiry that:
    - investigates a contemporary phenomenon within its real-life context, especially when
    - the boundaries between the phenomenon and context are not clearly evident.
2. The case study inquiry:
    - copes with the technically distinctive situation in which there will be many more variables of interest than data point, as one result
    - relies on multiple sources of evidence, with data needing to converge in a triangulating fashion, and as another result
    - benefits from the prior development of theoretical propositions to guide data collection and analysis.

We performed the following steps (according to Yin [14]):

- Design the case study (Section 6.2)
- Develop the case study (Section 6.3)
- Analyse the data gathered (Section 6.4)
- Develop conclusions and recommendations (Chapter 7)

The type of the case study developed in this research is an *exploratory case study* [14], in order to answer the research questions described in 6.1.Introduction. It is exploratory because the phenomenon of the co-existence of agile practices and SPI standards has not been investigated before (the problem is in a preliminary stage). The results of the case study will then be compared to the findings from the literature review done previously (in Chapter 4) or the framework in Chapter 5. In addition, the results of the case study can provide better understanding into the given situation of adopting and adapting agile practices into a mature organisation.

In what follows, we first briefly introduce the company that was our case study organisation. After that we describe the design of the case study, which consists of descriptions of types of questions that were used and the units of analysis. Next, we introduce the method of data collection, and then we analyse the data collected to draw conclusions.

Topicus

We conducted the case study in Topicus, an innovative ICT service provider located in Deventer. The company traditionally develops web-based solutions for Dutch marketplace [75]. Now Topicus specializes in supply chain management and realisation of SaaS (Software as a Service) in applications for the specific areas Finance, Healthcare and Education.

Topicus employs principles of ASD across their whole organisation. That is, all their projects are based on agile methods.

6.2. Case study design

This section explains the important components that should be designed in the design stage [14] to be ready for data collection as well as defining the criteria for interpreting the findings.

6.2.1. Questionnaire

According to Stewart and Cash ([70], chapter 4), questions can be divided in three categories: 1) *open* and *closed*, 2) *primary* or *secondary*, and 3) *neutral* or *leading*.

The questions employed in the questionnaire in this case study were mostly *open, primary* and *neutral*.

- Open questions: the questionnaire includes mostly the open questions so that the researcher can obtain coverage in sufficient depth and breadth. Closed questions, though only a few, were not used alone, because they are very restrictive in what the research is looking for, and the answers are almost given as mentioned in the questions. The few closed question were always followed by “how” and “why” questions.
- Primary questions are listed in Appendix D, but the interviewer also came up with some secondary questions while conducting the interviews in order to obtain more information from interviewees (the information was related to the answers of the primary questions).
- Neutral questions were developed to give the interviewees the freedom to give the answer according to what they think or know (or want). Leading questions were not used since they could influence the results of the case study.

For the detailed questions in the questionnaire, please see Appendix D (Case study guide).

6.2.2. Units of analysis

This section discussed the units or conceptual factors for analysing the data of the case study. Since the most important factors for software development success are people, process, and tools, the conceptual units (listed in Table 6.1) that related to software and those factors are analysed. A good understanding of these factors and units can help software companies in planning process improvement actions as they are getting more mature.

Units	**Description**
Customer satisfaction	Which aspects customers are satisfied with or they want to be improved?
Efficiency of communication	How efficient is the communication between project members.
Level of documentation	What kind of documentation is available in the company (e.g., does the requirements document or technical specification exists and is it available for the developers and testers?)
Selection of tools	How strongly is the selected tool reflected on software development practices?
Level of Involvement	How well customer understand the agile ways and participate in the software process, as well as how strong weight of the word customer has
Software quality	The density of defects in software program.
Value creation	Customer's perceived trade-off between benefits (what the customer receives) versus sacrifices (what he or she gives up)
Waste reduction	How many obstacles are removed from the quick delivery path of software by improving the development process?
Software process	Phases in the product life-cycle through which the software is being produced
Agile practice	Concrete agile activities and work-products that a method defines to be (regularly) used in the process.

Table 6.1. Units of analysis.

6.3. Data collection

The objective of this section is to discuss the method of data collection used for this case study as well as the procedures of collecting data.

6.3.1. The interviews

A theme-based interview is the main data collection method. There are three main themes organized on the case study guide (the questionnaire):

- The company profile: about the departments and projects in the company, and how it becomes more mature.
- The company software processes: about agile processes and practices, as well as those practices and processes the company has to implement to be compliant with the level of maturity.
- The company's perception on key issues: how the company and its customers think about software quality, customer satisfaction, value creation etc.

6.3.2. Selecting the interviewees

The interviewees who were selected are the people with knowledge and experience in software development from projects in Topicus. Their knowledge is gained through their experience and their roles in the projects which are:

- Systems Analyst: professionals with this role analyse functions and features needed for each increment in each iteration or each cycle of the software development process. A system analyst also interacts with customer to elicit the requirements as well as helping developers during system development.
- Project manager: the person who takes care of the planning, project deliverables, process, quality assurance, etc.
- Developer: is a person who codes, programmes, refactors, or implements the tasks in each iteration into a working system.
- Service manager: the main responsibility of a service manager is to communicate with customers during the development for the features, functions, and the changes in the requirements.

The interviewees in this case study will be referred to as SA1, SA2…SAn for their roles as systems analyst; PM1, PM2…PMn for the roles of project manager; Dev1, Dev2…Devn for developers and SM1, SM2…SMn for service managers.

6.3.3. The interview procedure

- The interviews are mainly conducted using face-to-face conversation. In addition, emails were also used to back-up for the issues which are not clearly obtained during the interviews or for those arise later from the interviewees' answers. Four face-to-face interviews and three emailing interviews have been conducted.

- The approximate time for each face-to-face interview is one hour.
- A digital recorder was used in each interview.

6.4. Data analysis

6.4.1. Analytic strategy

There must be an analytical strategy that will lead to conclusions. Yin [14] (pg. 111-115) presented three strategies for general use. One is to rely on theoretical propositions of the study, and then to analyse the evidence based on those propositions. The other technique is to develop a case description, which would be the framework for organizing the case study. The third technique is to set up a framework based on rival explanations, which can be related to the first strategy in that the original theoretical propositions might have included rival hypotheses.

The general strategy used in this research is the second one, where the descriptions are those resulted from the literature review in the previous steps (see Chapters 3-4-5, with the definitions, descriptions and results) and are reflected in the case study guide (the questionnaire). This also means that the analysis is concurrent with the data collection phase rather than really subsequent to it.

In order to produce an analysis of the highest quality, four principles suggested by Yin [14] (pg. 137) are followed:

- Show that the analysis relied on all the relevant evidence.
- Include all major rival interpretations in the analysis
- Address the most significant aspects of the case study
- Use the researcher's prior, expert knowledge to further the analysis.

6.4.2. The analysis and results

This section aims to analyse the data gathered in the data collection step based on the units of analysis and draw conclusions that will form our main results. But first we investigate the maturity of the company, as the context in which those units operate.

6.4.2.1. Topicus' level of maturity

Since Topicus does not base their software development processes on CMM or ISO-based framework, its maturity level is investigated through interviewees' opinion and certain characteristics of being mature according to CMM-SW. CMM-CW Level 2 requires the ability to repeat the processes and practices, and the Level 3 states that a well-defined process can be characterized as including readiness criteria, inputs/outputs, standards, the procedures for performing the work, and completion criteria [21].

Below, we present the findings that suggest the level of maturity in our case study organisation. We first state the theme that came out of the interviews and then we provide statements of our interviewees and our reasoning on them.

a. General point of view: the organisation is pretty mature.

SA1: "*Matching to CMM the maturity of Topicus is more or less between level 2 and level 3*".

PM2: "*I think we are very mature… In the range from one to five, our maturity can be level 3*".

First, we must note that we analyse maturity from interviewees' general point of view. The first evidence is the fact that Topicus had traditionally small projects in the past, with five or six people each, thus it was suitable for choosing agile strategy for software development (SA1). However, the projects have been getting bigger and bigger, that makes the company continuously improve the processes. Hence, the company processes are not chaotic or ad hoc anymore. Second, in the Finance department, the mortgage domain (which is the dominant domain) is pretty mature with a lot of domain knowledge (PM1). From the development side, the department has good developers and delivers good software in reasonable amount of time. Moreover, they tend to put functionalities into components, and those components are reused by and shared with other projects (Dev1). This practice is also applied in other departments (Healthcare, Education). The existing components are believed to be mature since they have been used and improved for years. From project management side, however, they believe that there is room for improvement, especially with the scalability (PM1: "*From project development side, there is room for improvement*"; SA1: "*We are quite mature, but we always want to be better"*).

b. The processes are quite repeatable.

In the interviews, the ability to repeat the processes is investigated. The interviewees indicated that from the high level view, most projects use the same agile process – Scrum, together with some XP-like practices. For example, they all have iterations (or Sprints/small cycles) of two or three weeks, they have Scrum meetings, and certain level of customer involvement etc. The general procedures for developing software are as following (PM1):

- Write a business case: even before starting a project
- Define what the customer really wants
- Estimate how long it will take, how many iterations and how long each iteration will take
- Start the project: continually refine that estimation. Always try to plan what is going to be in each iteration, how much effort, which people are allocated, what test scripts are used etc.
- Repeat that for each iteration

In each iteration, the following activities were performed:

- Analysis and design: functional design
- Implementation: implement the items in functional design.
- Test: customer is also involved; the customer is especially responsible for acceptance tests
- Fix, and deliver a bug-fixed release.

All the phases of the development process for new projects are based on the experience within similar projects or phases in the past (PM1, Dev1, PM2). That allows Topicus to repeat successful practices developed in earlier projects, although the specific processes implemented by the projects may differ to some extent.

What is more, the company always learns and improves their processes in order to be better. This is shown from the culture of Topicus: "*We encourage ideas and opinions from new and junior people… we do not completely rely on what old and senior people for what we should do. Some steps already went well in the past, but if a new person thinks and convinces others doing like this or this would be better, then we try the new way*" – SA1. This is also evidence for *self-organizing teams*, in which each team determines the best way to handle work.

c. The processes are not so well-defined

We observe how defined the processes are. The first evidence is the adoption of Scrum in most projects in Topicus, which is claimed by all the interviewees. As mentioned in the discussion on Scrum (section 3.5), although the standard Scrum [42] have been maintained and refined during the last decade, it is still based on the concept that the software development process is not a defined process, but a black-box one with complex input/output transformations. From this point of view, the main development process in Topicus (Scrum) is quite disciplined but not so well-defined.

The second evidence is that the inputs/outputs of the processes are not always formally defined, and sometimes depends on customers (to the extent of "*how formal the customer wants to be*" – PM1). For instance, for one customer the input for the design step is formal requirements, and the output is the design document; but for another customer, the input can be only few notes or words on white-board (and Topicus still makes the output a design document).

We also get an evidence that for each process and its steps, "*people in each project have the high-level ideas and concepts of what needs to be done*" – SA1. They usually do not have written-down documents which are clear, quantifiable or measurable about the inputs and the outputs. The inputs and outputs are sometimes changed.

There is also evidence that in some projects, people try to accomplish the clear inputs and outputs of the steps/phases during the development process (PM2). One example is the project that has to build a proof of concept in which everything should be as clearly designed as possible, and also prototypes are used.

d. A lot of standards for software development process are used

Topicus uses UML in the design phase, standard coding method or coding conventions for programming phase, unit testing and TMAP (a test tool) for the test phase in which they are trying to automate the process (in Finance department the degree of automation in testing is 10%, in Education department it is 30-40%).

Moreover, concerning developing environments, Topicus Finance uses Visual Studio.NET and SQL server, whereas in Topicus Healthcare and Education, Java and Oracle are the most commonly used. They also use HL7 (a layer in OSI model for exchanging messages [80]) in certain projects that need a proof of concept.

e. There are completion criteria for the development process

Completion criteria are those that show when the process is completely done. They are especially important for agile processes, since agile processes are iterative and always responsive to changes.

There are completion criteria for each phase of a process in Topicus. The design step is done when all design documents are created, the implementation is done when all items (in design documents or in Product Backlog) are implemented, and the release is done when it passes all the test cases. Moreover, those criteria are kept consistent during the whole process. The similar idea is also obtained from other interviewees:

SA1: "*Overall a product is considered finished when all the features and functions designed are implemented and the customer tests it themselves and accept it… After the customer tests a release, they normally have a test report of change request. If there is no more change request then the product is done*".

PM2: "*a release is only considered done when unit tests in the entire system run at 100%*".

To sum up, it is possible to conclude that the maturity level of the software development processes in Topicus is in between level 2 and 3 of CMM-SW. The processes are repeatable and have certain characteristics of being defined (but not all their parts are defined). As the company has always been agile, we can consider its development side maturity as *agile maturity*. Comparing to the scaling categories in Agile Scaling Model [69], the practices used in Topicus is reaching *disciplined agile delivery*, in which people already look beyond ASD and consider the full complexities of solution delivery to employ disciplined processes. That is, the development teams are self-organizing within an appropriate governance framework.

6.4.2.2. Topicus's software development processes

Most projects in Topicus use Scrum process to develop software. Matching the general procedures in section 6.4.2.1 to the names used in Scrum yields the most commonly used process in Topicus (PM1) (in fact, some adoptions of Scrum do not use exactly the names from standard Scrum, but still keep similar meanings and usage):

- Step 1: Plan a business case: the outcome is the Product Backlog which is regularly updated. The items in that backlog are prioritized and effort for each work item is estimated to implement.
- Step 2: Take the highest priority items into a Sprint Backlog. Divide the items in Sprint Backlog into requirements and set the goal for the next Sprint.
- Step 3: Each requirement goes through an iterative Sprint consisting of analysis, design, implementation, testing (unit and acceptance) and delivery phases. Finish the Sprint, obtain new product increment (with new functionality or feature); and update the Product Backlog (in step 1).
- Repeat steps 2 and 3. If there are no more requirements in Sprint Backlog, then go to Step 4.
- Step 4: Integrate and carry out the system testing. Release the final version and create necessary documentation.

At implementation level, since there are differences from project to project, the Scrum practices used in different projects are different. Some projects use Scrum board, daily stand-up meetings, but some do not. In addition, the use of Scrum roles also varies: there is a Scrum Master in most of the projects, but sometimes there is not (Dev1).

6.4.2.3. Agile practices used in Topicus

From the data collected, there are only a few practices that needed to be further recognized and named according to their usage in Topicus. In other words, the data analysis revealed practices that were not explicitly named by the interviewees. The other agile practices are easily named by the interviewees since they are very standard and popular practices. One of those practices is *on-site customer* but in the case of Topicus it is usually not the same as the standard on-site customer (which says that the customer works with the development organisation at the same physical location 100% of the time). Topicus's clients are intensively involved in the development process, but that is not completely "on-site" (SM1). Agile teams work closely with them, but not on a daily basis. More often than not, employees of the clients come to Topicus for couple of days and vice versa. In addition, as also mentioned in section 6.4.2.1, and this also applies to Topicus, that the level of commitment to Scrum differs from project to project, and not all Scrum practices and roles are employed (Dev1).

PM1: "*We use Scrum… XP a little bit less… we have Scrum meetings, user stories board, short cycles, acceptance test, and simple design*".

SA1: "*We use Scrum and XP… we do have Scrum meetings, we use small releases, testing, collective ownership, continuous integration, on-site customer, coding standards*"; "*Design* *and development was not completely 'on-site', because the members of both organisations weren't at the same physical location 100% of the time, but often, employees of the client came to us for a (couple of) day(s) or vice versa.*"

Dev1: "*We use Scrum so we have most of Scrum roles like ScrumMaster, Product Owner, Team, UI designer, tester, programmer… but the level of commitment to Scrum is different by projects… we usually have Scrum stand-up meeting…*"; "*we do have small releases, simple design, pair programming but sometimes it does not necessarily mean coding… what is metaphor?*".

There are two practices that need to be recognized by further analysis. The first one is *metaphor*. By definition, metaphor is the "shared story" that guides all development with how the whole system works [2, 45]. They use the metaphor practice since the professionals in Topicus always have the high-level ideas and concepts of what needs to be done. For example, if a module from existing components, is inconsistent with the ideas or concepts (the shared story), then they know the module is wrong or not appropriate. In addition, often a metaphor boils down to a system of names. The names provide a vocabulary for elements in the system and help to define the relationships. Topicus does use coding conventions that include the standard use of names.

The second practice is *open workspace*. This practice is recognized by description from interviewees and from the researchers' own observation. A team works together in an open room where there are tables set up with workstations on them. The programmers are in a position to communicate intensely: each has the opportunity to hear when another is in trouble and each knows the state of the work of the other.

For the purpose of clarity, Table 6.2 lists the practices used in the company. It is noteworthy that many XP practices are used together with Scrum practices thus creates a set of hybrid practices in Topicus.

Scrum practices	**XP practices**
(Daily stand-up) Scrum meetings	Small releases
Scrum boards	Testing (acceptance testing, unit testing, system testing)
Burn-down graphs	Collective ownership
Work-breakdown structure	Continuous integration
User stories board	User stories
Iteration Demo (Sprint review)	Coding standards

	Metaphor
	Simple design
	Pair programming
	(Semi-)On-site customer
	Open workspace

Table 6.2. Agile practices used in Topicus

6.4.2.4. Topicus's maturity standards

The maturity of the whole Topicus is achieved by adopting SAS 70 (Statement on Auditing Standards No. 70 [77]) and ITIL (Information Technology Infrastructure Library [78]) standards. These standards are currently used for service support and service delivery, and are not so connected to the software development side.

a. SAS 70

SAS 70 is the auditing standard for service organisation processes (e.g., incident management, change management). However, at the time of conducting this case study, there is little connection between SAS 70 and the software development processes in Topicus, as the SA1 says:

"*Currently SAS 70 does not directly apply to the software development. We do not have any rules that tell developers that they should write code in a certain way or should use a specific programming language. We have internal programming guidelines and best practices that handle that part of development quality*."

This, however, does not mean that having standardized service organisation processes does not improve software quality. For example, in its SAS 70 framework it is written that no part of code can be released without testing. As a result, every change that is implemented is tested within a few days. Then, developers get feedback on their implementation and they are confronted with bugs that are found. This improves quality because they are more directly aware of the result of their work.

Moreover, since with SAS 70 processes are written down, they "*lower the risk that certain steps are forgotten when doing a certain task*" (SA1). This means there is a lower chance of something going wrong or something unexpected happening.

b. ITIL processes

ITIL is a set of concepts and best practices for information technology services management (ITSM), information technology (IT) development and IT operations that is widely used in IT. Because of its broad acceptance, Topicus uses some of the ITIL processes as core processes for its internal SAS 70 framework.

Topicus's SAS 70 control framework is based on the service support and service delivery part of ITIL.

6.4.2.5. The relationship between Agile and maturity standards in Topicus

Topicus maintains both agile approaches and compliance with the standards in parallel. That is, they keep agile for the development side and sell the products and services as standardized solutions while deploying SAS and ITIL processes for service management side. This way, they satisfy more customers who require certification and thus acquire more projects. The company keeps the development side agile since Agile is what they do best and they are already familiar with agile processes and practices.

Moreover, one of the main principles of SAS 70 is "say what you do, do what you say". That is, even though it has no pre-described requirements for agile processes and practices from the development side, to get compliant with the certification, development organisations still need to create documentation of how they are applying agile (agile process documents). Therefore, although having a standardized service organisation processes helps improve software quality, there can be more overhead because more tests and measures need to be done to ensure the quality, and also more documentation for the processes is needed.

That also means that the agile processes need certain adaptation. As described in section 6.4.2.2 about the adoption of Scrum process in Topicus, the Scrum process in the company includes the step of creating the necessary documentation. Now it is probably the case that they have to create more extensive testing documents and process documents, instead of only creating as little documentation as possible.

6.4.2.6. The adaptation of agile practices to become more mature

While Topicus was getting more mature, it experienced a continuous adaptation of the agile practices which have been employed. First, the practices have evolved due to the internal motivation within the company itself. In the beginning the company started with using a specific set of agile toolkits, but later they found that some practices definitely worked for them, some did not, and others needed adaptation. Furthermore, the practices have always been adapted to fit to the customers. Agile methods and practices used in Topicus are customer-dependent, since not every customer is equally experienced in working with agile approaches. For some practices, Topicus tries to convince the customer to work in the way which Topicus is already familiar with. For others, however, they need to adapt themselves. For example, Topicus usually convinces customers to have frequent communication, face-to-face conversation and close interaction with them. The ideal case is to have an *on-site customer*. Other examples of practices that they tries to convince customers are *small releases* and *testing*, which require frequent customer involvement. However, Topicus sometimes have to adapt the practice on-site customer as well, due to the fact not every customer is familiar with Agile or experienced in software development.

Therefore, Topicus bases on specific customer characteristics to understand which practice is better to convince, which is better to adapt themselves. For example, Topicus can convince customers who are inexperienced (or immature) in software development, by insisting that their agile way is what they do best. For customers who have large and hierarchical decision making scheme that slow down communication, Topicus has to adjust the practices that require interaction with customers (see Section 6.4.2.14 for more detail).

6.4.2.7. Customer satisfaction

There are three aspects of the software development process in Topicus that their customers are most satisfied with. First of all is the speed of deliveries. Topicus produces working software on a regular basis, typically in the context of short, time-boxed iterations or cycles. This allows customers to be able to quickly see how the software is working. Second, customers can be really involved in seeing and testing so that they have good idea of what Topicus is doing, and clear view of how they do it. They can provide feedback and ask Topicus to go to the later cycle in a different way. Moreover, for inexperienced customers, Topicus starts building working software (based on their experience and domain knowledge) even when the customers does not know what they want. Topicus shows them the piece of working software and it gets easier for both parties. The third aspect is the good amount of functionalities that Topicus delivers because of having good domain knowledge in specific areas, the reuse of proven existing components and the short cycles.

Together with getting more mature, Topicus provides broader spectrum of functionalities as well as responding quicker to customers' change and feedback (for example, Dev1 said *"Client satisfaction improves because we respond quicker"*). Moreover, they are creating more and better documentation of the agile processes (in order to be compliant with the certification SAS 70 as well) to show their confidence and expertise to customers ("*Documentation is now better*" – PM2). Therefore, they achieve higher client satisfaction.

6.4.2.8. Tracking and measuring software quality

From the development perspective, Topicus considers high quality software as having a small number of defects (all interviewees talked about bug tracking system and testing, when they were asked about software quality). This fits with the common understanding about software quality that we experience from literature review. However, they do not measure software quality as the number of defects per thousand lines of code. Instead, they often have ideas (based on their experience) about how many bugs can exist in projects of certain size. Therefore, in some projects, they just focus on five to ten bugs (PM2).

As far as software quality tracking is concerned, all departments in Topicus use the open-source bugs tracking system Mantis which clarifies three main measures: severity, priority of a bug, and the number of bugs. *Severity* is about the risk when something is not in the system, or not correctly in the system; *Priority* is about how fast or urgent they need to fix the bug. Since Mantis is an open-source system, each project or department can change its features to fit with their situation. In general, each project defines a set of different levels of severity and priority: high, medium, low. That is, high severity means it is a very serious defect, or low priority implies that the bug can be delayed to be fixed until the next iterations.

The number of bugs is known when testers, developers and customers use and report on the system. The project manager also keeps track of software quality by an account on Mantis. The developers carry out the tracking by 1) doing code review based on coding conventions, and 2) a number of unit testing. Unit testing is also supported by a tool called jUnit.

When it comes to the customer, it is an on-site acceptance test on Mantis. Topicus usually has fall-back scenario in that they can go back to the previous step or iteration if the software does not pass the test. However, the quality of existing components (for re-using) is normally proven in advance.

While getting more mature, software developed by Topicus also gets higher quality:

PM1: "*Nowadays our software normally has less defects than before, because of using standard components*".

PM2: "*We have better quality software, much less bugs*".

SA1: "*In our SAS 70 framework we say that no part of code can be released without testing. As a result every change that is implemented is tested within a few days. As a result our developers get feedback on their implementation and their confronted with bugs that are found. This improves quality because we are more directly aware of the result of our work*".

Dev1: "*There are fewer defects than before, because problems are identified and solved earlier, and we use common components*".

To sum up, Topicus has a technically good bug tracking system. Mantis and jUnit are very important tools for testing in the company. However, they have not really measured software quality and this makes the estimation of quality improvement difficult, although they assert that software quality do increase over time as the company is getting more mature.

6.4.2.9. Waste reduction

Since most projects in Topicus are fixed-price contracts, waste reduction in the company is mainly gained from the reduction in development time. Being more mature with agile, Topicus does reduce the development time.

PM1: "*We reduce waste by reusing components and domain knowledge…sometimes something planned to be built in iteration X needs to be redone in iteration Y, which might be considered waste. But to the extent the customer can quickly see the function and test it functioning, so we gain better customer relationship and customer satisfaction. So it's a kind of trade-off*"; "*I think the development time get shorter and we reduce cost as well*".

Dev1: "*We got shorter development time because being agile we skip some documentation, and also we build what customers need, not all what they want*".

PM2: "*Time gets shorter and costs are reduced because we don't have to fix bugs or problems in the later phases*".

But there are also other ideas about the total time of a project as Topicus is creating more and more documentation:

PM2: "*We gain shorter development time but the total time from the beginning to delivery does not reduce much because we make more documentation*";

SA1: "*Getting more mature and implementing SAS 70 doesn't automatically reduce time but in the long run it will… so we spend time now to gain time in the future*".

6.4.2.10. Value creation to customer

Topicus's value creation to customer is generally perceived the same as user satisfaction. That is, due to the agile approach and the re-use of existing components, Topicus delivers software in a small amount of time, thus makes time to market of the customer shorter. Moreover, since the development processes employed in Topicus are flexible and adapting to customer in each project, the customer works with Topicus more easily. And last, also due to having a great deal of domain knowledge and existing components, Topicus does not only develop software faster, but also provides more useful functionality.

From this perspective, value creation partially drives user satisfaction when affecting time to market, functionalities and usefulness of the software. That is the reason why Topicus' professionals also think that by getting more mature, they create more value to customers as well.

6.4.2.11. Knowledge transfer

a. Level of documentation

Topicus' professionals use various documents during the software development, and they are all available to developers and other members of a project. First, they have coding convention which tells developers the coding standards of how to name functions and variables etc. This document is continuously improved in each department as people gain more experience. Moreover, there are some templates of design documents (functional design), test plan, test cases, change request document, and process documents (or process guidelines, e.g. PRINCE2 [76]). For these templates, Topicus' people do not follow strictly the conventions, but "learning and doing by example" (SA1). For example, some templates say they should follow the steps described by certain chapters, but in specific situations they do not really need some chapters and they can neglect them. Furthermore, in some projects, domain specific documents (e.g., about HL7) are used.

Most of the documents are in Microsoft Office formats. For example, all written-down documents are in .DOC file.

While coding convention is consistently used in most projects, the number of other documents such as design documents, change request and the degree of their usage vary by projects. For testing, there is usually not much documentation since project members put "test report" on the bug tracking system Mantis.

b. Efficiency of communication

Since Topicus' customers are not always on-site, it is difficult to communicate as effectively as the agile approach promises. There is evidence that Topicus' professionals have to wait a long time for customers' feedback and decisions on implementing some features, and sometimes they just continue with the iteration if they think the features could be and should be approved.

The efficiency of communication between all members in a project might be fostered by the Topicus' culture. The culture enables and encourages communication as well as speaking out people's ideas to make the common work better. In that culture, the organisation does not have a hierarchical structure; normally there is only one official group manager and everybody else is equal. Some standard may say do certain tasks in some ways, but if a developer thinks differently, he can convince others to do differently.

6.4.2.12. The selection of tools

As mentioned in section 3.2.1, ASD values individuals and interactions over processes and tools, but this does not mean people doing agile do not value tools. In fact, Topicus uses a variety of tools for their software development.

There are pieces of evidence showing that the use of these tools is very important for projects in Topicus, especially in programming and testing phases. First of all, for developing environment, they use IDEs such as Visual Studio.NET, Eclipse together with certain plug-ins (e.g, PMD) to check code, help developers make code and to improve coding quality. Second, for testing phase, they have jUnit for unit testing and Mantis for tracking bugs. Moreover, it is the case that continuous integration is tool-dependent. The use of tools is even more essential for big projects (of 30-40 people) (SA1). Without tools, it is almost impossible to develop software in Topicus (Dev1).

In contrast, for team cooperation, in most projects there is no tool and the communication and cooperation between members are mostly done on social basis. In some cases, however, people use Sub-version SVN (which is open source software with Apache license that helps developers share and update on the same code).

For the planning and management side, other "simpler" tools are used such as Scrum boards, burn-down graphs, work-breakdown structure, or user stories board. These tools are really "agile" tools (they are agile practices as well) and quite independent on the development tools. Therefore, some specific projects may not use some of them.

In summary, the tools used in Topicus are always project-based, not organisation-based. Moreover, while development tools and testing tools are strongly reflected on the development practices, the planning and management tools are more flexibly used.

6.4.2.13. Level of involvement of the client

The conceptual factor *Level of involvement* refers to how familiar a customer is with agile ways of developing software and how well he or she participates in the software process, as well as how strong is the weight of the word the customer. In the case of Topicus, this factor is project context-dependent and customer-dependent.

First, how well Topicus' customers understand the agile approach depends on each customer. There are customers who are inexperienced in software development or are not used to agile approaches, whereas there are those who are very familiar with working in agile or semi-agile ways. For those who are inexperienced, Topicus tries to convince them to follow the way the company works by saying firmly that the way is what they do best. The same strategy can be applied for customers who are experienced (in software development) but not familiar with agile processes. Still for some clients Topicus has to adapt their process to fit with the client's organisation's way of working. The weight of customers' words, therefore, is quite strong. But this might also be interpreted as the flexibility of agile methods in Topicus.

Second, the effectiveness of customer participation in Topicus' software process is also customer-dependent. In all projects, the analyst (from Topicus) spends a lot of time to: 1) talk with the customer about what the product needs or what is going to be; 2) show the progress of the iterations. There is a lot of interaction between the customer and the development team but the customer is not always really part of the team itself. There is evidence that in a small project (of 6 or 7 people), the client was very much involved in the development process. The project team consisted not only of people from Topicus, but also from the client who are designers, programmers and testers. In that case, design and development was not completely "on-site" (the members of the two organisations were not at the same physical location 100% of the time), but often, employees of the client came to Topicus for a couple of days or vice versa. On the other hand, there was a big project of 30 people (in 18 months) in which the communication between Topicus and the customer was slowed down due to the customer's large and hierarchical organisation. In this case, decisions had to be made by multiple people, thus making the feedback from the customer slow and the communication was not very effective.

6.4.2.14. The project-based degree of commitment to agile

From the previous analysis, we conclude that it is the case that Topicus uses different agile practices for different projects, depending on the "characteristics" of customers and projects. That means, agile practices used in Topicus are project-based, and for certain projects it is easier to use specific agile methods and practices, while for others it is more difficult or not possible to use the practices.

We have obtained evidence about project-based data concerning the degree of application of agile practices. They are project-based data since they focus more on characteristics and context factors of a project, which are *the type of the product, type of the contract, length of the project, team size, how critical is the application*, and *how formal are the requirements from customers*. Boehm and Turner [59] also define the five critical factors that describe a project environment and help determine method balance between agile and plan-driven, they are: size (*team size*), criticality (*how critical is the application*), culture (may cover *how formal are the requirements from customer*), dynamism (the change rates, may cover *specific customer character*), and personnel (expertise of the team, according to Cockburn's three level of software understanding [4]).

The data currently cover two types of projects: small projects and big projects. The small project in this case (project A) has the ideal size for mainstream agile processes and practices (fewer than 10 developers). The two projects both belonged to Topicus Finance, which has a maturity level in between CMM levels 2 and 3 (as analysed above). Both projects were successful. Table 6.3 lists the characteristics of the two projects.

Characteristics	**Project A**	**Project B**
Type of product	Application supports mortgage process.	Application supports mortgage process.
Customer	Banking sector	Banking sector
Length	3 months	18 months
Type of contract	Fixed price	Fixed price
How critical was the application	Heavily regulated by laws and regulations	Quality aspects as availability are of critical importance
Team size	Small, 6 or 7 team members, 12 members during the peak of the development	30 people
Specific customer character	Used to work in (semi-) agile way	Large and hierarchical decision making scheme.
Formal specifications	There are not many formal specifications from the customer.	More than 2000 requirements specified
Communication with customer	Close collaboration with customer; customer involvement and quick feedback	Slow communication; slow decision making and feedback from customer
Developing	Scrum iterative development	Scrum iterative development

method		
Ease of agile adoption	Easy to adopt agile strategy for the entire project	Difficult to make a truly agile development

Table 6.3. Project-based characteristics.

As can be seen from this specific project-based data, getting customers' participation, interaction and involvement is one of the biggest challenges that software companies face in order to adopt successfully agile approaches. This is shown by the practices adopted in the two projects listed in Table 6.4. For the larger project (B), some practices are used to the lesser extent as those in the small project A. For example, PM1 said: "*We try to use as much open workspace as possible. However, due to the fact that the total number of project members was rather large and we had the restrictions of our current office, we had to divide people over more rooms*".

Some practices are used in Topicus but they were used in other projects. There is no evidence that they are used in these two projects (A and B). As we can see in Table 6.4, those practices are *daily stand-up Scrum meetings*, *burn-down graph*, *metaphor*, and *pair programming*.

Practices	**Project A**	**Project B**
Scrum boards	U	UL
Work-breakdown structure	U	U
Daily stand-up Scrum meetings		
User stories	U	U
Burn-down graph		
Iteration demo/ Sprint review	U	
Small releases	U	
Testing	U	U
Collective ownership	U	
Continuous integration	U	U
Coding standards	U	U
Metaphor		

Simple design	U	
Pair programming		
(semi) on-site customer	U	
Open workspace	U	UL

Table 6.4. Practices used in the two projects. (U = used; UL = used but to a lesser extent as the small project)

6.4.2.15. What needs to be improved?

As mentioned earlier, Topicus need to improve its management side to be more scalable (PM1). Topicus' projects get bigger and bigger, leads to bigger products and probably more bugs during the development. Moreover they need to improve the level of agility in big projects where there are many people involved. Currently Topicus' staff think they should base on experience for the improvement:

SA1: "*we will find our own way to do agile better with larger project group, for example with 30-40 people… base on experience we'll see what works and what doesn't so that we can improve our practices*".

Topicus' professionals also want to improve other areas. Some want more detailed and specific design (PM2). The testing process also needs to be enhanced by more automated testing and the testing should be more complete. From a developer's point of view (Dev1), there are needs for more support from boss and more experienced people. Topicus also needs to keep regular meetings (for example, Scrum meetings) in every project as it is a good practice but it is less used in some projects in the company. In addition, the developer wants the company to keep employing new tools to support developing software.

To sum up, we can see some desired improvements are about keeping the best agile practices or agile philosophy (e.g., more experienced people, more meetings, more scalable agile, automated testing), whereas some are about more defined and formal ways of developing software (e.g., more detailed and specific design, more supporting tools). This might match with the fact that Topicus' agility is getting more mature, and another fact that Topicus implements more regulatory and formal processes (ITIL and SAS 70) for the phases after a product is finished. This all helps improve the quality of Topicus's products and services, and gain more customers as well as their satisfaction.

6.5. Conclusions

After analysing the data, we summarize the results in forms of answers to the research questions:

RQ1: Which software development and management processes do organisations have in order to be compliant with their current level of maturity?
Since Topicus gets more mature in agile manners, there is no specific process that is implemented to help Topicus gets compliant with its maturity level.

RQ2: Which agile processes or practices do they have?
Topicus has an adaptation of Scrum process, and it uses mostly Scrum practices together with XP practices. We observed six main Scrum practices and ten basic XP practices in the company. Topicus does not employ some XP practices such as *planning game* and *40-hour week* (or *sustainable pace,* see Table 3.3). We did not find evidence for many Scrum events and practices such as *Sprint planning*, *Sprint review*, *Sprint retrospective*, *just-in-time planning*, and *just-in-time design.*

RQ3: How processes in RQ1 and RQ2 co-exist or impact each other?
There is no specific process needed for Topicus' "real" maturity, but Topicus certifies for an auditing standard SAS 70 for its service management side. This makes the company to implement ITIL processes of the service support and service delivery parts. SAS 70 has certain impacts on software development processes in Topicus due to its requirements. SAS 70 requires that no part of code can be released without testing, thus Topicus has to test each change in each iteration more frequently. The company also has to make more documentation about its agile development process for auditing.

Chapter 7: Disscussion and recommendations

The goals of this chapter is to discuss certain issues related to the results of the case study, as well as to provide some recommendations for companies like Topicus which also have agile strategies and consider implementing a standard to prove their maturity level.

7.1. Discussion

According to the results presented in section 6.4, we think it is possible to make the conclusion that good process development approach in Topicus has led to a quite disciplined "Way-of-Working" (WoW [47]). Topicus' staff expresses their pride on the WoW and always keep focusing on good process to continuously enhance the WoW.

Our case study results indicated the following issues that need to be discussed:

1. Although Topicus is getting more mature not by really implementing a process maturity model, its mature development processes have the same effects on the software product as with the SPI-based processes. The first effect is the increase in quality due to fewer defects and less rework in the later phases of a process. Higher quality is also gained in the support of the auditing standard for the service side of the company. The second effect is waste reduction due to the decrease in development time and due to less rework as well. Moreover, getting more mature, Topicus also creates more value to customers and make them more satisfied.

2. The case study results show that agile processes and practices in Topicus are quite disciplined, and they meet most requirements for CMM level 2 and level 3. Table 7.1 matches the agile practices used in Topicus against the CMM-SW key process areas. We make the note that the matter that we can establish this matching does not mean that the organisation meets all the goals specified by the key process areas addressed by agile practices. In other words, it cannot be concluded that the use of all those agile practices and Topicus processes automatically lead the organisation to CMM levels 2 or 3 maturity. Topicus would need extra effort and some more traditional or plan-driven practices. Moreover, the adequacy of the practices depends on the context where they are used (see more in section 6.4.2.14). For large projects, or projects operating in more challenging environments, some additional or alterative practices maybe needed.

The matching (Table 7.1) can only tell us that Topicus can continually use and improve the practices to gradually meet the KPAs' goals. However, it is really difficult for Topicus to be certified for higher maturity according to defined process methodologies, while they continue keeping agile practices that best fit the organisation. This is firstly because with agile ways of developing software, a lot of practices and tasks cannot be (partially or fully) described. Since they are tacit knowledge, they cannot be fully documented into process documents. Moreover, software is considered to be a complex system, as there is no way in which one set of controls or standards can be put in place in order to provide a predictable or repeatable desired outcome. Therefore, there is a need for a good agile maturity model to assess the maturity level of Topicus. However, if people in the company consider the ultimate goal of software process improvement is to understand and repeat successes and to understand and avoid failures [72], they can propose their own software process improvement approach base on what they are currently doing: set the phases of the development process for new projects based on their experience (currently they do not have any roadmap for improving the process).

CMM Level	**KPAs**	**Topicus' agile practices**
Level 5: Optimizing	Defect prevention	Continuous integration
	Technology change management	Not applied
	Process change management	Not applied
Level 4: Managed	Quantitative process management	Not applied
	Software quality management	Not applied
Level 3: Defined	Organisation process focus	Scrum meetings, Iteration Demo (Sprint review)
	Organisation process definition	Metaphor
	Training programme	Not applied
	Integrated software management	Not applied
	Software product engineering	Metaphor, Simple design, Testing (unit and functional), Coding standards
	Intergroup coordination	(semi) On-site customer, Pair programming, open workspace.
	Peer reviews	Pair programming
Level 2: Repeatable	Requirements management	(semi) On-site customer Continuous integration Product Backlog Sprint Backlog

	Software project planning	Small releases Work-breakdown structure Product Backlog Sprint Backlog
	Software project tracking and oversight	Small releases Scrum boards, User stories board, Work-breakdown structure, Burn-down graphs
	Software subcontract management	Not applied
	Software quality assurance	Pair programming Testing (unit testing, system testing)
	Software configuration management	Collective ownership
Level 1: Initial	[No key area]	[Processes and practices in Topicus are not ad hoc anymore.]

Table 7.1. Topicus' agile practices and CMM-SW KPAs.

3. The results of the project-based degree of commitment to agile once again confirm that context factors (in this case they are project context factors) do favour or impede the adoption of agile practices. Since many agile methods (e.g., Scrum) do not prescribe any specific engineering practices, the agile practices can be used flexibly depending on specific customer character, the project size, the type of the outcome product, or its critical level in practice etc. However, they still have to remain their basic meanings as well as the common and fundamental agile principles. In some context, it is hard or even impossible to apply agile principles. The result (section 6.4.2.14) confirms that the bigger project (size) the harder to use Agile, and that it is easier to adopt agile practices in the culture where people feel comfortable and empowered by having many degrees of freedom (project A, the customer is familiar with doing Agile) than in a culture where people have their roles defined by clear policies and procedures (the hierarchical culture in the case of project B).

4. As projects in Topicus are getting bigger and bigger, Topicus also face bigger problem of scalability. Firstly, Topicus is scaling up from project focus to domain or industry focus, due to the fact that in finance sector, the company is developing more and more software for back-end processes (from the start, the software focused on the request stage, that processed from banks' customers search for information until banks generate offers to customers). Secondly, and obviously the company is scaling up in team size. In the beginning, the project

team size is usually fewer than 10 developers, and now there are more and more projects with 30-40 people. Mainstream agile processes work very well for smaller teams (less than 10-15 people); but as team size grows, face-to-face strategies start to get more difficult to apply, and more paper-based work need to be included to manage and organize the team. That means certain plan-driven process aspects need to be integrated in order to overcome the challenge of scalability.

5. There is evidence that in some cases Topicus' customers have to make trade-off decisions between quality, functionality and time. For example, when a customer wants more features or more iteration for a product, then the customer has to either give Topicus more time or reduce features in other iterations. It should be better for Topicus to protect schedule commitments despite changes, because agile principles are about increasing team efficiency, lowering development costs and faster time to market. Based on our observations, we believe that most customers frequently change their requirements after product design has begun, but they expect the software to be delivered without delay.

6. It is noteworthy that by continually improving the coding standards in coding convention document, Topicus' professionals help the refactoring practice as well, in changing and adapting the code to make it cleaner and more maintainable without changing functionality. However, refactoring as an agile practice is not yet explicitly observed in Topicus. This might be because the company focuses more on adopting Scrum method rather than on specific agile practices described in XP method.

7.2. What do we really need to do?

This section aims to give some recommendations or guidelines for companies like Topicus which has been pursuing agile strategies and becoming more and more mature.

In general, for the purpose of continuous process improvement, organisations that embraced Agile should learn more about CMMI and understand CMMI as it was intended. It is true that they can improve software development process and organisational maturity based on their experience. However, agile people should also become familiar with CMMI practices that complement agile practices addressing non-functional requirements, product architecture, measurement and analysis, risk management, and organisational learning etc. Nowadays, it might be the case that most agile development teams do not follow a CMMI-compliant process (in a survey that was conducted in 2009 [63], 78% of agile people answered "No" to the question "Does your agile development team follow a CMMI-compliant process?").

There are several other issues that should also be taken into consideration when we want to implement a process maturity model into an agile environment.

7.2.1. Justifying if a certification is needed

We believe that each company should have a business justification in order to pursue a certification of high maturity such as ISO or CMM. The justification should be in the form of a business case and should at least estimate the cost, effort (personnel) and time needed to change (improve) all required processes and create necessary documentation, as well as the number of requests from current customers and also from potential customers for the certification. The implementation and adaptation of the two approaches (Agile and process maturity model) should be kept in track with respect to the organisation's effective achievement of the board's strategic goals and actual performance at delivering value to customers.

After a company has decided to implement a SPI standard or plan-driven methodology, it might have to deal with extremists in both areas who refuse to open to the change. Because changing processes involves changing culture, this is where it is critical to get everyone thinking about the changes in a positive ways.

7.2.2. Keep gaining trust

One of the agile principles is building projects around motivated individuals with high level of trust among them. To be more mature and disciplined, the issue of trust is even more important. We think that it is more difficult for each development team to instil trust into customer than gaining trust among developers. From agile people side, they should better use the idea of working software and customer participation to gain trust [50], and that is also what Topicus' professionals use (Topicus also rely on their clients' word of mouth and their reputation to gain trust from new and potential customers). In general, they can use their track record, the systems they have developed and the expertise of their people to demonstrate capability. If a new customer does not trust the developers, the team can get their trust by performing the following tasks [65]:

- Meet with the customer
- Explain the processes
- Demonstrate some preliminary system
- Capture the customer's feedback and share with the rest of the team
- Copy the urgent, high-value stories into the product backlog
- Plan and complete the sprint
- Go back to the customer to show them the new features/functionality.

From the process people side who directly deal with assuring a process maturity model, especially when they are already certified for high maturity (e.g., a CMMI), they can count on their process maturity (the certification) to provide confidence in their work.

It is also recommended that in order to be successful with software projects in agile ways of working, software companies need to take the right to say "No" to customers who do not want to work according to the values and principles as defined by the Manifesto for ASD [1], or those who do not play their roles as required by the companies' process [47, 55].

7.2.3. Strong management support for Agility-Maturity

There is always the need for the visible support and commitment from senior management, especially for enterprise or project-wide adoption of agile methods as teams become more self-organized by using agile practices, for instance, Scrum practices. Management should know the issues involved so that they can provide strategies and resources, as well as to satisfy ISO and CMM requirements such as:

- Management commitment to the defined software development process
- Setting up and staffing an independent quality assurance group
- Reviewing the implementation of the various defined procedures etc.

The issue of senior management involvement is stated by Vriens [53] as a major challenge to overcome in order to be certified for a CMM.

7.2.4. Certain level of training needed

Companies should train people to perform agile methods, because the CMM requires people to be trained [47, 55]. Some customers could be invited to join training courses. The training should make sure all the parties involved trust and appreciate each other, and most importantly, ensure that they have clear communication. It should also deal with sharing tacit knowledge (for example, by mentoring). Besides being knowledgeable (on discipline and agility), development team members need to be highly motivated to work and to share their knowledge. The training can also help build culture of excellence, which gathers all good skilled people (e.g., developers) who have the entire caring attitude to customers; who always ask for what customers really want to achieve, and who always try to improve their team processes.

7.2.5. Concern about violating some agile principles

The first agile principle that might be violated by getting compliant with a SPI standard is that of "*working software over comprehensive documentation*" [56]. The solution is to provide just enough documentation to be a useful reference to the developers (for training new developers as well) and to help with the enforcement of existing processes. This includes documentation describing the company's agile development methodologies and their benefits, and documented procedures describing good practices on how to perform key development processes. While writing those documents, we should take in to consideration the standard's guidelines (CMM-SW, ISO 9001) so that the documents can be good enough to be used as proof of conformance and can be reviewed as part of the standard's verification and validation.

The key factor in successful certification without a negative impact on agility is to base quality management on the natural outputs of the development process (i.e., working software, customer satisfaction) rather than on artifacts produced solely for audit processes (i.e. documentation, reports).

7.2.6. Scaling agile methods

As mentioned in literature, and as we also observe in the case of Topicus, there is a problem of scaling agile methods when the size of the team or the scope of the project is large. The problem can be partially solved by implementing certain traditional or plan-driven practices. In addition, agile experts can scale up projects by providing explicit and early attention to non-functional requirements and product architecture [68], and by introducing a top layer of coordination (e.g., "Scrum of Scrums" [68, 69, 79]). Scrum of Scrums is a technique that is used to organize a big project team into multiple teams, and make up a Scrum team of representatives from each of the teams. Cross-team issues and dependencies then are resolved through that Scrum team.

7.3. Threats to validity

There are four main threats to validity for our conclusions of our case study results, which are the following: 1) have we understood SPI-based standards CMM and ISO 9001?, 2) have we understood the maturity of Topicus?, 3) have we understood agile development in general and Scrum and XP in special?, and 4) have we fully examined all agile methods and practices in the company? They are discussed in more detail below.

7.3.1. Have we understood the SPI standards?

In our study, we focus on two standards ISO 9001 and CMM-SW. We base on the standards themselves together with their guidelines, and then we go through each whole standard. In addition, we coordinate our own examination with the review of other relevant literature. Therefore, we are confident that we assessed all relevant issues concerning the standards and agile methodologies.

However, we lack personal practical experience with certifying or auditing a standard like ISO 9001 or CMM-SW. We assume therefore that different auditors may have different set of viewpoints for what they think is acceptable and that their viewpoints may be different from those expressed in this thesis. Also, agile methods are mostly not strictly defined, and they are employed flexibly, leads to a possibility that what we found acceptable for the matching between agile practices and SPI-based standards – for example, the XP planning game – may not be accepted as planning process for CMM-SW by some auditors.

7.3.2. Have we understood the maturity of Topicus?

With all the data we collected, we are confident in the validity of our assessment about Topicus' maturity. However, since there is still not a business justification from Topicus for a CMM certification, our assessment only serves as a reference for the purpose of this study. There is also a possibility that different auditors may assess Topicus' level of maturity differently when Topicus wants to become ISO 9001 or CMM certified.

A threat to validity might be poses by that the author of this thesis misinterpreted the information provided by the interviewees regarding the level of maturity in Topicus. We think however that this threat is reduced to a minimum because the three supervisors of the author reviewed the results and triangulated them with their own knowledge of the Topicus organisation. We make the note that the supervisors had many previous master projects in Topicus and they were confident in their own understanding of the results.

7.3.3. Have we understood agile development?

As we already mentioned, ASD is not an exact defined methodology. There are many agile methods that vary, and in this research we focus on XP and Scrum. However, they are all based on the common principles described in chapter 3, and each method is also based on its few principles and basic practices. In the literature review we mostly draw conclusions on results of other published studies. However, we try to apply the common principles and the meanings of the basic practices in the case study, and a certain level of subjective interpretation is inevitable.

7.3.4. Have we examined all agile methods and practices in the company?

As stated in the previous section, we try to apply the common and fundamental principles of agile methodology as well as the meanings of the basic practices of each agile method into the case study.

However, we interviewed a number of professionals at Topicus and not the entire Topicus organisation, and we are clear on that these professionals' project experiences may not be representative for the whole Topicus organisation or for the whole set of agile projects at Topicus. We deliberately chose professionals with diverse background that run agile processes in different application areas (e.g. financial services applications, education support applications). Drawing on our observations, we think that the conclusions would hold for other project delivery units at Topicus in contexts similar to those of the professionals that we interviewed. We consider our findings indicative as the professionals were chosen for their typicality in the organisations. Moreover, the author of the thesis triangulated the information regarding the agile processes which he receives from the interviews with information about the company, which he receives from other sources (e.g. his supervisors who had been exposed in their prior projects to the Topicus organisation and some of the interviewed professionals). This increased the confidence that the examined agile processes are enough for us to get findings that indicate the ways through which Topicus manages to increase its level of maturity while remaining agile.

7.3.5. Our claims to validity

Based on the discussion above, we believe that our observations and conclusions are relevant for the topic and defendable while reasoning about the possible threats to validity.

7.3.5.1. Internal validity

We conclude that our analysis provides sufficient evidence to support the claims in our conclusions because it systematically followed the analysis principles ([14]). First of all, in the analysis we attended to all evidence. We have gotten as much relevant evidence as was available, and our interpretations account for all of this evidence. Second, the analysis addresses all alternative explanations (rival interpretations) for one or more of our findings. A few rival interpretations will be investigated in the future studies. Moreover, our analysis addresses the most significant aspects of our study, and that reduces the possibility that the main issue is being avoided because of possibly negative findings. And lastly, we take advantages of prior knowledge which are the results of our previous studies and publications.

7.5.3.2. External validity

The outcomes can also be generalized to:

- Other SPI-related standards such as ISO 9001, ISO 15504 (SPICE), TickIT. This research only examines with CMM-SW and some relationships with ISO 9001.
- Other agile methods together with Scrum and XP such as Crystal, Feature-Driven Development (FDD)

In conclusion, we claim that our conclusions regarding SPI standards and ASD will be valid for a wide range of software companies which follow agile strategy and auditing companies (for auditing the maturity of the company and the standards' compliance).

Chapter 8: Conclusions

This study provided evidence that a software company delivering solutions exclusively through ASD and APM practices, can - with a little extra effort, achieve also a high maturity certification as CMM-SW Level 2 and 3. The company can also get other certifications such as ISO 9001 or SPICE. It also shows that while working on the compliance to a maturity standard, *it is not necessary* to define strictly procedures, processes or to create huge amount of documentation. As Watts Humphrey states in the preface of [21]: "you need to look at what *people* are actually doing". In most jobs the work practices (how it is really done) are different from the work procedures (how it is supposed to be done). Therefore, a combination of good engineering practices and good enough documentation would be ideal.

Though having certain limitations, this study has implications both for research and practice.

8.1. Practical implications

From practical perspective, the study has the following implications. First, it provides a foundation for a framework that can be used to help organisations embrace both high maturity standards and agile practices. Also, practitioners can use the conclusions in this report as one of the sources for establishing concrete guidelines which show how agile methods can be enhanced to fully cover all the process areas that are not in conflict.

Second, the results of this study can be valuable for other companies that aim at improving their software development processes as well as increasing their maturity. For example, an agile project manager that needs to convince his/her senior management that agile and process maturity concepts can co-exist and benefit from each other, can use this study as an example of how it all happen. Also, other agile professional who might fear that the achievement of a higher maturity level would come at the expense of the Agile manifesto, could use our study and based on our results, they might evaluate the extent to which their fear is justified in their organisations.

Third, in our literature review, we studied a number of case studies (conducted in other authors' research) and based on this, we derived the conceptual models that state the possible relationships between the concepts of maturity level, waste, value creation, client satisfaction and software quality. These relationships are hypotheses that practitioners might be curious about regarding whether or not project experiences in their own organisations confirm or disconfirm them.

8.2. Theoretical implications

We have listed most of the important topic-relevant results in the literature review part of this research. This includes the more comprehensive (than done in other relevant studies) matching between XP and Scrum practices and CMM process areas, the degree of compatibility between process maturity model and the agile methods, and how the two approaches are implemented and adapted together. This has an implication for other researchers who would want to develop a conceptual model or a framework for implementing a process maturity model using a combination of agile methods.

In addition, as agile methods are very good approaches to the issue of how to implement a SPI standard and model into an organisation, it is suggested from this research that by adopting agile practices, organisations fit more KPAs of CMM. Also, in the same company, the level of CMM-based processes adopting agile development methods would be higher than the level of processes without adopting agile methods. These “hypotheses” could be evaluated in another scope of research.

8.3. Limitations of this study

This study has also several limitations. First of all, only XP and Scrum have been discussed. To make our results more general, further agile methods such as Crystal methods and Feature Driven Development [45] should be analysed as well.

Second, this study reviews a diversity of empirical research studies dealing with merging Agile and CMM. Most of them are mappings where specific practices of some process areas of CMM are merged with Scrum or XP. This might lead to a false impression of full understanding of the subject because that ignores higher level of agility (agile methods and processes) and other components of CMM (for example, generic practices).

And last, the case study is only performed in a company which has been agile by design (and since it was founded) and is on its pursuit to higher maturity. Also, in the case study we have tried to look at different sub-organisations and projects in the company, but the data collected are limited. This might have restricted the full observation of the real-life phenomena as well as the generation of exact conclusions.

8.4. Implications for further research

First of all, further case studies can investigate how companies can become more mature while they are inherently being agile. Additional case studies might include companies that are actually CMM certified, or companies that successfully scale up as their projects get bigger.

Second, we could also collect additional data about specific projects in Topicus or other software companies adopting agile practices, to further examine how context factors impede or support the adoption of agile project management and ASD practices.

Another direction for further study is on the model for agile maturity, at least in a specific context like that of the case study in Topicus. The maturity model classifies levels of maturity in adoption of agile practices by a certain organisation, and shows areas of interest that the organisation should address in order to continuously improve. The model could learn from the road to being more mature of Topicus (even not pre-defined or written down), and the agile scaling model [69] would play as a conceptual framework or a classification scheme for that.

References:

[1] **Agile Alliance** (2001). Manifesto for agile software development. [Online] Retrieved 19 March 2010. Available at: http://agilemanifesto.org .

[2] **Beck K.** (2000). Extreme programming explained: embrace change. Addison-Wesley. (reading book)

[3] **Boehm B., Turner R.** (2004). Balancing Agility and Discipline - A guide for the perplexed. Addison-Wesley. (reading book)

[4] **Cockburn A.** (2006). Agile software development: the cooperative game. Addison-Wesley. Second ed. (reading book)

[5] **Jarvis B., Gristock S.** (2005). Extreme Programming, Six Sigma & CMMI – How they can work together, a JP Morgan Chase Study.

[6] **Koch A.S.** (2005). Agile Software Development - Evaluating the Methods for Your Organisation. Artech House, Boston. (reading book).

[7] **Lami G., Falcini F.** (2009). Is ISO/IEC 15504 applicable to agile methods? *XP 2009*. LNBIP 31, pp. 130-135.

[8] **McMichael B., Lombardi M.** (2007). ISO 9001 and Agile Development, IEEE Agile.

[9] **Paulk M.** (2001). XP from a CMM perspective. IEEE Software, 18 (6), pp. 19-26.

[10] **Pikkarainen M., Mantyniemi A.** (2006). An approach for using CMMI in Agile software development assessments: experiences from three case studies. SPICE 2006 conference.

[11] **Santana C., Gusmao C., Soares L., Pinheiro C., Maciel T., Vasconcelos A., Rouiller A.** (2009). Agile software development and CMMI: What we do not know about dancing with elephants. XP 2009, LNBIP 31, pp. 124-129.

[12] **Stalhane T., Hanssen G. K.** (2008). The application of ISO 9001 to Agile Software Development. *PROFES 2008*, pp. 371-385.

[13] **Sutherland J., Jakobsen C., Johnson K.** (2007). Scrum and CMMI Level 5: The Magic Potion for Code Warriors. In: Proceedings of the Agile Development Conference, pp. 466 – 471.

[14] **Yin R. K.** (2002). Case Study Research: Design and Methods, Applied Social Research Method Series, 3rd ed., Sage Publications.

[15] **Wright G.** (2003). Achieving ISO 9001 certification for XP Company. XP/Agile Universe, LNCS 2753, pp. 43-50.

[16] **Herbsleb J., Zubrow D., Goldenson D., Hayes W., Paulk M.** (1997). Software quality and the Capability Maturity Model. Communications of the ACM, 40(6), pp. 30-40.

[17] **Krishnan, M.S**. (1996). Cost and Quality Considerations in Software Product Management. Dissertation, Graduate School of Industrial Administration, Carnegie Mellon University.

[18] **Agrawal M., Chari K.** (2007). Software effort, quality, and cycle time: a study of CMM level 5 projects. IEEE Transactions on software engineering, 33(3), pp. 145-156.

[19] **Diaz M., Sligo J.** (1997). How Software Process Improvement helped Motorola. IEEE Software, pp. 75-81.

[20] **Harter D.E., Krishnan M.S., Slaughter S.A.** (2000). Effects of process maturity on quality, cycle time, and effort in software product development. Management Science, 46 (4), pp. 451-466.

[21] **Paulk. M.C., Weber C.V., Curtis B., Chrissis M.B.** (1995). The Capability Maturity Model: Guidelines for improving the software process. Addison-Wesley Publishing Company (Reading book)

[22] **Racheva Z., Daneva M., Sikkel K.** (2009). Value creation by agile projects: methodology or mystery? 10th International Conference on Product-Focused Software Process Improvement (PROFES 2009), pp. 141-155.

[23] **Pitterman B.** (2000). Telcordia technologies: the journey to high maturity. IEEE Software, pp. 89-96.

[24] **Conradi R., Fuggetta A.** (2002). Improving Software Process Improvement. IEEE Software, pp. 92-98.

[25] **Suryn W.** (2008). Software quality engineering: the leverage for gaining maturity. Pp. 33-55.

[26] **Parzinger M. J., Nath R.** (2000). A study of relationships between total quality management implementation factors and software quality. Total quality management, 11 (3), pp. 353-371.

[27] **Kauppinen M., Savolainen J., Lehtola L., Komssi M., Tohonen H., Davis A.** (2009). From feature development to customer value creation. IEEE International requirements engineering conference 17th, pp. 275-280.

[28] **Little T.** (2004) Value creation and capture: a model of software development process. IEEE Software, 21 (3), pp. 48-53.

[29] **Barney S., Aurum A., Wohlin C.** (2008). A product management challenge: Creating software product value through requirements selection. Journal of Systems Architecture, 54, pp. 576-593.

[30] **Favaro J.** (2005). That elusive business value: some lessons from the top. Lecture notes in computer science, 3556, pp.199.

[31] **Woodruff R.** (1997). Customer Value: The Next Source for Competitive Advantage, Journal of the Academy of Marketing Science, 25 (2), pp. 139-153.

[32] **Johnson, W.C., Weinstein, A.** (2004). Superior Customer Value in the New Economy: Concepts and Cases, CRC Press, Boca Raton, FL. (Reading book)

[33] **Jones, M.A., Taylor, V.A., Becherer, R.C., Halstead, D. (2003)**. The impact of instruction understanding on satisfaction and switching intentions. Journal of Consumer Satisfaction, Dissatisfaction and Complaining Behavior, pp. 10-18.

[34] **Ranaweera, C., Praghu, J.** (2003). On the relative importance of customer satisfaction and trust as determinants of customer retention and positive word of mouth. Journal of Targeting, Measurement and Analysis for Marketing, 12 (1), pp. 82-90.

[35] **Paulk M.C.** (1995). How ISO 9001 compares with the CMM. IEEE Software, 12 (1), pp. 74-83.

[36] **Clark B.K.** (2000). Quantifying the effects of Process Improvement on Effort. IEEE Software, 17, pp. 65-70.

[37] **Alegría J.A.H., Bastarrica M.C.** (2006). Implementing CMMI using a combination of Agile methods. Clei electronic journal, 9 (1), paper 7.

[38] **Schwaber K.** (2004). Agile Project Management with Scrum, Microsoft.

[39] **Marcal A.S.C., Freitas B.C.C., Soares F.S.F., Belchior A.D.** (2007). Mapping CMMI project management process areas to SCRUM Practices. Proc. SEW 2007 31st, pp. 13-22.

[40] **Martin R.C., Newkirk J.W., Koss R.S.** (2003). Agile software development: Principles, patterns, and practices. Prentice Hall. (Reading book)

[41] **Ambler S.W.** (2002). Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process. Wiley Computer Publishing.

[42] **Schwaber K.** (1995). The Scrum development process. In OOPSLA '95 Workshop on Business Object Design and Implementation, Austin, Texas, USA, October 1995. ACM Press.

[43] **Jakobsen C., Sutherland J.** (2009). Scrum and CMMI – Going from Good to Great. Are you ready-ready to be done-done? In Agile 2009, Chicago.

[44] **Schwaber K., Sutherland J.** (2010). Scrum guide. [Online] PDF guide. Available at: http://www.scrum.org/scrumguides/ .

[45] **Abrahamsson P., Salo O., Ronkainen J., Warsta J.** (2002). Agile Software Development methods: Review and Analysis. VTT Publications (478), pp. 3-107.

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

[47] **Vriens C., Barto R.** (2008). 7 years of agile management. Proc - Agile 2008 conference, pp. 390-394.

[48] **Webster J., Watson R.T.** (2002). Analysing the past to prepare for the future: Writing a literature review. MIS Quarterly, 26 (2), pp. xiii – xxiii.

[49] **Kitchenham B.** (2004). Procedures for performing systematic reviews. NICTA Technical Report.

[50] **Turner R., Jain A.** (2002). Agile meets CMMI: Culture Clash or Common Cause? XP/Agile Universe 2002, LNCS 2418, pp. 153-165.

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

[52] **Huo M., Verner J., Zhu L., Babar M.A.** (2004). Software quality and Agile methods. Proc of COMPSAC'04, pp. 520-525.

[53] **Vriens C.** (2003). Cerifying for CMM Level 2 and ISO9001 with XP@Scrum. Proc of the Conference on Agile Development. Pp. 120-125.

[54] **Kahkonen T., Abrahamsson P.** (2004). Achieving CMMI Level 2 with enhanced Extreme Programming approach. PROFES 2004, LNCS 3009, pp. 378-392.

[55] **Bos E., Vriens C.** (2004). An Agile CMM. XP/Agile Universe 2004, LNCS 3134, pp. 129-138.

[56] **Paulk M.C.** (2002). Agile methodologies and process discipline. Carnegie Mellon University.

[57] **Cao L., Mohan K., Xu P., Ramesh B.** (2009). A framework for adopting agile development methodologies. European Journal of Information Systems, 18, pp. 332-343.

[58] **Beck K.** (1999). Embracing change with Extreme programming. Computer, 32 (10), pp. 70-77.

[59] **Boehm B., Turner R.** (2003). Using risk to balance Agile and Plan-driven methods. Computer, 36 (6), pp. 57-66.

[60] **Mann C., Maurer F.** (2005). A case study on the impact of Scrum on overtime and customer satisfaction. Proc of Agile conference (ADC 05), pp. 70-79.

[61] **Ramesh B., Cao L., Mohan K., Xu P.** (2006). Can distributed software development be agile? Communications of the ACM, 49 (10), pp. 41-46.

[62] **CMMI Product Team** (2006). CMMI for Development, Version 1.2, CMU/SEI-2006-TR-08. Software Engineering Institute, Carnegie Mellon University.

[63] **Ambler S.** (2009). July 2009 Agile Practices Survey. [Online] Retrieved May 2010. Available at: http://www.ambysoft.com/surveys/practices2009.html

[64] **Ambler S.** (2008). February 2008 Agile Adoption Rate Survey. [Online] Retrieved May 2010. Available at: http://www.ambysoft.com/surveys/agileFebruary2008.html

[65] **Upender B.** (2005). Staying Agile in Government Software Projects. Proc. of the Agile Development Conference (ADC'05), pp. 153-159.

[66] **Wailgum T.** (2007). From Here to Agility. [Online] Retrieved June 2010. CIO, available at: http://www.cio.com.au/article/193933/from_here_agility/

[67] **Scrum Community** (2009). Firms Using Scrum. [Online] Retrieved 10 July 2010. Available at: http://scrumcommunity.pbworks.com/Firms-Using-Scrum

[68] **Glazer H., Dalton J., Anderson D., Konrad M., Shrum S.** (2008). CMMI or Agile: Why not embrace both!. Technical Note CMU/SEI-2008-TN-003. Software Engineering Institute, Carnegie Mellon University.

[69] **Ambler S.W.** (2009). The agile scaling model (ASM): Adapting agile methods for complex environments. IBM Rational software.

[70] **Stewart C.J, Cash W.B.** (1997). Interviewing principles and practices. McGraw-Hill Higher Education, 8th Ed.

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

[72] **Nawrocki J.R., Walter B., Wojciechowski A.** (2001). Toward maturity model for eXtreme Programming. IEEE Computer Society, In 27th Euromicro Conference (Warsaw, Poland, December 2001), pp. 233-239.

[73] **Alleman G.** (2004). Blending agile development methods with CMMI. Cutter IT Journal, 17 (6), pp. 5-15.

[74] **Nonaka I.** (1991). The knowledge-creating company. Harvard Business Review, July-August 2007, pp. 162-171.

[75] **Topicus** (2010). Topicus website homepage. [Online] Retrieved 18 August 2010. Available at: http://www.topicus.nl/

[76] **Office of Government Commerce** (2005). Managing successful Projects with PRINCE2. The Stationery Office Books. 5th revised edition.

[77] **AICPA** (2004). SAS No. 70 and Service Organizations. American Institute of CPAs.

[78] **Office of Government Commerce** (2001). Service Delivery. IT infrastructure library. The Stationery Office.

[79] **Larman C., Vodde B.** (2008). Scaling Lean & Agile Development: Thinking and Organizational tools for large-scale Scrum. Addison-Wesley Professionals, 1st Edition.

[80] **Health Level Seven International** (2010). Official website. [Online] Retrieved 23 July 2010. Available at: http://www.hl7.org/

Appendix A. A quick overview of Extreme Programming

A quick overview of Extreme Programming (XP) practices [2, 3, 9, 40, 45]

Why eXtreme Programming?

Extreme Programming provides a systems perspective on programming. The reason for the "extreme" in the name is that XP takes commonsense principles and practices to extreme levels. For example, if code reviews are good, we will review code all the time (pair programming); or if architecture is important, we will work defining and refining the architecture all the time (see more in table A).

Commonsense	**XP extreme**	**XP implementation practice**
Code reviews	Review code all the time.	Pair programming.
Testing	Test all the time, even by customers.	Unit testing, functional testing.
Design	Make design part of everybody's daily business.	Refactoring
Simplicity	Always work with the simplest design that supports the system's current functionality.	The simplest thing that could possibly work
Architecture	Everybody works to refine the architecture all the time.	Metaphor
Integration testing	Integrate and test several times a day.	Continuous integration
Short iterations	Make iterations extremely short – seconds, minutes, and hours rather than weeks, months and years.	Planning game

Table A. The "extreme" in Extreme Programming [9].

III. **The four software project variables**: cost, time, quality, and scope. XP assumes that these four variables can be controlled.

- *Scope* is what the system does and what should be done in the future.
- *Quality* is about the measures used to fulfill the customer needs and expectations, for example reliability, correctness, security…

- *Cost* is the variable of people (personnel), technical equipment, and space that needed for the project.
- *Time* is about the duration of the project or how long the project takes to be done.

  For a project, if three of these variables are chosen, we can figure out the forth. Normally customers define scope and quality (to the extent of testing the product) in cooperation with the development team if necessary, and then the cost of resources is given. The development team then figures out how long the project will take.

IV. **The four values:** the fundamental XP values are [3]

- *Communication.* As most projects fail because of poor communication (between developers and developers, customer and developers, manager and customer, manager and developers), following XP means implementing practices that force communication in a positive fashion.
- *Simplicity.* Develop the simplest product that meets customer's needs. Make the code simple and easy to understand.
- *Feedback.* Developers and customers must obtain and value feedback from the system, and from each other. Firstly, feedback is obtained when customers make use stories (the features) and developers make the estimation (about how long will it take to develop as specific story) for each story. Secondly, customers also participate in functional or acceptance testing where they can see if the application meets their needs, and they get feedback from the system. And lastly, developers make unit testing to detect the bugs in the systems, to fix them and improve the quality of the product. They obtain the feedback of how the application is going with this unit testing.
- *Courage.* Be prepared to make hard decisions that support the other principles and practices. The courage is built on well understanding of the system due to simplicity, feedback within the project and good channels of communication.

V. The principles:

1. Fundamental principles
   - *Rapid feedback* through mechanisms such as unit and functional testing.
   - *Assume simplicity*, achieved by constant focus on minimalist solutions.
   - *Incremental change*: dealing with requirements changes through an iterative life-cycle with short cycles. Any problem is solved with a series of the smallest changes that make a difference.

- *Embracing change*: the best strategy is the one that preserves the most options while actually solving your most pressing problem.
- *Quality work*: developers continually write unit tests that must run flawlessly; customers write acceptance tests to demonstrate that functions are finished.

2. Less central principles. They will help us decide better what to do in specific situations:
    - Teach learning
    - Small initial investment
    - Play to win
    - Concrete experiments
    - Open, honest communication
    - Work with people's instincts, not against them
    - Accepted responsibility
    - Local adaptation
    - Travel light
    - Honest measurement

VI. **The four basic activities**:

- coding
- testing (unit and functional)
- listening
- designing

VII. The practices:

Beck's [2] explanation of XP can be understood in terms of a project management strategy and a project development strategy, each of which comprises a set of practices.

Project management strategy – 4 practices: On-site customer, small release, metaphor, and planning game.

Project development strategy – 8 practices: 40-hour week, simple design, testing, refactoring, pair programming, collective ownership, continuous integration, and coding standards.

Following the short description of all practices:

- *The planning game*: quickly determine the scope of the next release by combining business priorities and technical estimates (based on how much work the developers were able to get done in the last iteration). As reality overtakes the plan, update the plan. The results are really short iterations (can be as short as hours) and frequent releases.

- *Small releases*: put a simple system into production quickly, and then release new versions on a very short cycle.
- *Metaphor*: Guide all development with a simple shared story of how the whole system works. That means keeping defining and refining the architecture.
- *Simple design*: The system should be designed as simply as possible at any given moment. Extra complexity is removed as soon as it is discovered
- *Testing*: Programmers continually write unit tests, which must run flawlessly for development to continue. Customers write tests demonstrating that features are finished (functional testing).
- *Refactoring*: This is about good design. Programmers restructure the system without changing its behavior to remove duplication, improve communication, simplify, or add flexibility [40].
- *Pair programming*: All production code is written with two programmers at one machine or workstation. This also means keeping good code (peer) reviews all the time. This dramatically increases the spread of knowledge through the team, as well as significantly reduces the defect rate [40].
- *Collective ownership*: Anyone can change any code anywhere in the system at any time.
- *Continuous integration*: integrate and build (and also test – integration test) the system many times a day, every time a task is completed.
- *40-hour week*: work no more than 40 hours a week as a rule. Never work overtime a second week in a row.
- *On-site customer*: include a real, live user on the team, available full-time to answer questions.
- *Coding standards*: Programmers write all code in accordance with rules emphasizing communication through the code.
- (additionally mentioned in [58]) *Just rules*: by being part of an Extreme team, you sign up to follow the rules. But they are just the rules. The team can change the rules at any time as long as they agree on how they will assess the effects of the change.

Figure A represents the twelve original XP practices and their relationships. As shown in the figure, any one practice does not stand well on its own, but require other practices to keep them in balance.

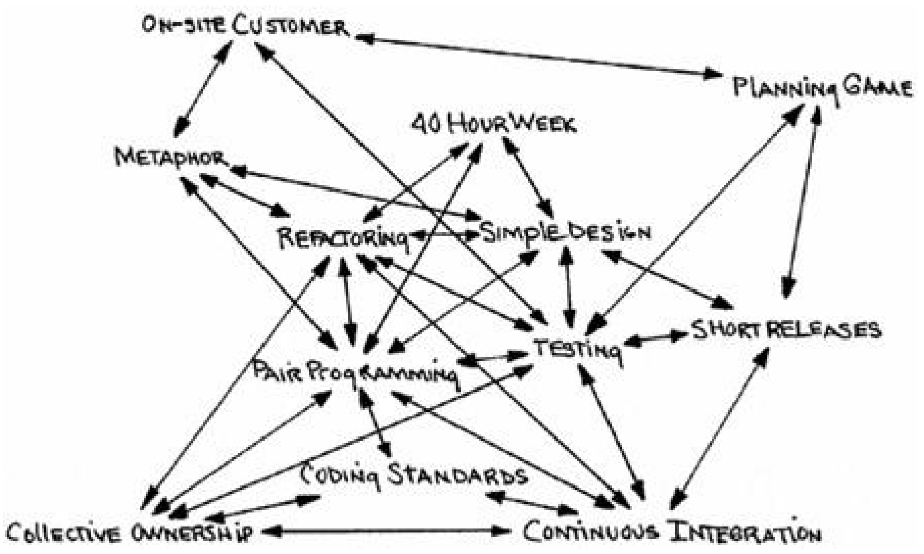

Figure A. The practices support each other. A line between two practices means that the two practices reinforce each other [2].

Appendix B. Scrum

Software development process is complicated with a lot of environmental changes, leads to the need of an approach that enables development teams to operate adaptively within a complex environment (therefore using imprecise processes). Producing well-arranged systems under chaotic circumstances requires maximum flexibility. Scrum uses control mechanisms to improve that desired flexibility.

The heart of Scrum is a Sprint – an iteratively fundamental development cycle (usually 30-day iteration). Sprint is nonlinear and flexible. The approach assumes that analysis, design and development processes in Sprint phase are unpredictable.

The Sprint phase is an empirical process, which is like a black box. Many of the processes in the Sprint phase are unidentified an uncontrolled, thus need external controls. These controls, including risk management, are put into each iteration of the Sprint phrase to avoid chaos while maximizing flexibility.

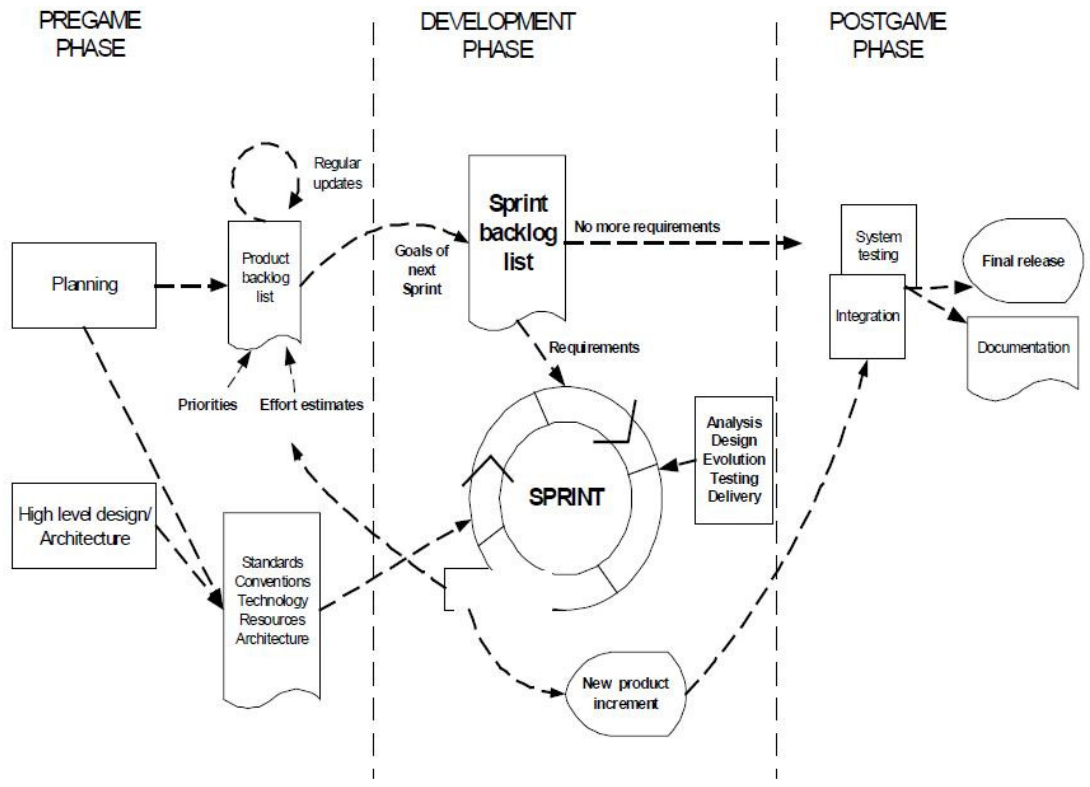

Figure B. Scrum Process.

To explain the Figure B, Sprint methodology goes through certain phases [42, 45]:

- The first phase, Pre-game, is where a Product Backlog is created (by the Product Owner working with the customer) with emerging and prioritized requirements for the product. This phase also includes a planning phase which defines the new release based on current known backlog, along with an estimate of its schedule and cost. Moreover, the architecture issues are taken into account as well with the design of how the backlog items will be implemented; this includes system architecture and high level design.

- Game or Development Sprints: the selected Product Backlog is put into Sprint Backlog for the development of new release functionality. There are multiple iterative development sprints (or cycles) that are used to evolve the system.

  What is more, each Sprint includes Daily Scrum Meeting in which each team member explains:

  - ✓ What he or she has accomplished since the last meeting?
  - ✓ What he or she is going to do before next meeting? and
  - ✓ What obstacles are in his or her way?

  Lastly, there is Sprint Review in the end of each Sprint when new functionality is demonstrated.

- Post-game or Closure: preparation for release, including final documentation, pre-release staged testing, and release.

Projects adopting Scrum have the following characteristics:

- Flexible deliverable: the content of the deliverable is influenced by the environment.
- Flexible schedule: the deliverable maybe required sooner or later than initially planned.
- Small teams: each team has the optimal size of 7 members, plus or minus 2. There may be multiple teams within a project [44].
- Frequent reviews: team progress is reviewed as frequently as environmental complexity and risk occurs (e.g., the regular Sprint Review). A functional executable must be prepared by each team for each review.
- Collaboration: intra and inter-collaboration is expected during the project

The name Scrum comes from the sport of rugby, in which scrum is a way to restart the game after an interruption, where the forwards of each side come together in a tight formation and struggle to gain possession of the ball when it is thrown in among them [42]. The Scrum methodology shares many characteristics with rugby:

- The context is set by playing field (environment) and rugby rules (controls).
- The primary cycle is moving to ball forward.
- Rugby evolved from breaking soccer rules – adapting to the environment.

- The game does not end until environment influences (business need, competition, functionality, and timetable)

Appendix C. The waterfall model

The waterfall model is one of the oldest software development process models, and is also the most mature one. The waterfall model divides the software development lifecycle into five distinct and linear stages (i.e., in practice the model can be followed in a linear way). However, some stages can be also overlapped. Iteration in an agile method can be treated as a miniature waterfall lifecycle.

The 5 distinct and linear stages:

- Requirements analysis and definition
- System and software design
- Implementation and unit testing
- Integration and system testing (verification)
- Operation and maintenance

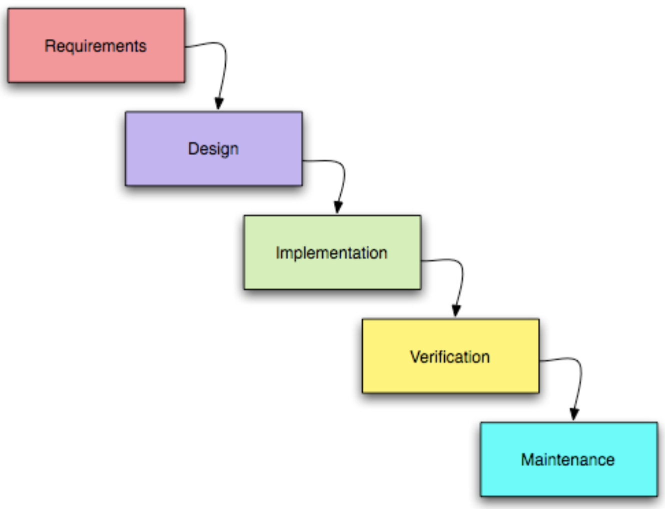

Figure C. The waterfall model

In principle, any stage should not start until the previous stage has finished and the results from the previous stage are approved.

The model has been very successful with large and complex systems. However, its linear nature has been its largest problem. The development process does not define how to respond to unexpected output from any of the intermediate process. It also shows the inflexibility in keeping up with the changing requirements, and highly bureaucratic processes irrespective of the nature and the size of projects [52].

SCRUM vs. Waterfall

Table C compares the primary characteristics between Scrum and Waterfall methodologies.

	Waterfall	**Scrum**
Defined processes	Required	In Planning & Closure steps only
Final product	Determined during planning phase	Set during project
Project cost	Determined during planning phase	Set during project
Completion date	Determined during planning phase	Set during project
Responsiveness to environment	In Planning phase only	Throughout the process
Team flexibility, creativity	Limited-cookbook approach	Unlimited during iterations
Knowledge transfer	Training prior to project	Teamwork during project
Probability of success	Low	High

Table C. Methodology comparison (adapted from Schwaber [42]).

Appendix D. Case study guide

Case study: Topicus

Introduction

The purpose of this case study is to explore what and how practitioners do. For Topicus, we study how it gets more mature while being agile.

Research questions

RQ1: Which software development and management processes do they have in order to be compliant with their current level of maturity?

- Process: the series of steps that the company takes in order to deliver a software product.
    - The process is complex
    - The process is changing?
- Since the company is doing agile way, and moreover they have achieved certain level of maturity, the processes, for example, can be XP or its variation (or something else similar) or Scrum or its variation.

RQ2: Which agile processes or practices do they have?

For example: Agile processes: XP, Scrum, etc.

Agile practices: on-site customer, pair programming, short iterations, etc.

RQ3: How processes in RQ1 and RQ2 co-exist or impact each other?

I. The interviewee

Could you please introduce a little bit about yourself? E.g. How many years of experience do you have? What is your role in the team/project?

II. Company profile

a. Industry

Could you introduce your company's operations?

(Can also be summarized from catalogs, company profiles, and other documents/articles)

Could you introduce your department operations and your project(s)?

b. Software development processes

- What is the level of maturity of your company?

  [You do not have to assess the maturity in some standard (for example CMM-based framework)]

    - Do all the projects in your company adopt the same processes of development software? (the ability to repeat the processes)
    - Does your process have clear inputs and outputs? Give some example.

        - o What are the standards (if any) used in your company development processes? (or a typical process)
        - o Does each process have the completion criteria (the criteria of when it is completely done)?
- What processes do you have to implement to get fit with that level of maturity?

c. When getting higher agile maturity:

- Which aspects of your development process or your software are your clients most satisfied with?
- What is the estimated degree of software quality (or the degree/the number of defects)?
    - ▪ Does your company track or measure software quality?
    - ▪ How do you (your company) track or measure software quality?

  *If they measure: what is the estimated number of defects per thousand lines of code? (or per function/ iteration?)

  (*depends on their way of measuring it*)
- How do you estimate the development effort/cycle time/costs in each normal project?
    - ▪ Cycle time measurement: the time span of iteration (or something similar in the case).

      Does your company keep the time constant (2-3 weeks) or it is subject to change?
    - ▪ Development effort: is the person-days (or months) required to develop a software product.
        - o Is the number of people in the project/ development team set constant in the beginning? How many are there?
        - o How do you estimate the effort for each iteration?
        - o *Are the estimation techniques really based on agile ways of doing software?*

          (Companies might estimate costs differently from using waterfall model to agile methods)
        - o Based the effort estimation, how are the required tasks assigned to the development team?
        - o How do you estimate the costs in money?

d. Are there issues where you think you could improve your customer satisfaction?

III. The company's software process

We heard that your company has achieved a certain level of maturity according to a Dutch standard.

a. What does it mean to have high maturity level according to Dutch standard?

b. What are the standard's requirements and principles?

c. What practices and activities were used?

  - What are the steps of a development process?
  - What are the roles of people in each project?

IV. Company's perception on key issues

a. Value creation

How do the processes in RQ1 & RQ2 create value?

- Do they make higher quality product? How?
- Do they make more satisfied clients? How?
- Do they reduce time/costs for development? How?

What do your clients think about value creation?

When do they consider that your software product/process creates their values?

b. Client satisfaction

What do you think makes your clients satisfied?

V. The adoption (or implementation/application) of agile methods

a. Agile practices and methods

What are the agile methods and practices used?

Are they (agile practices) intended to be matched with maturity of your company? Or do they contradict in some way?

What kind of documentation do you use? (Coding conventions/ standards; test case documents?...)

What is the format of the documents? (.doc, .txt)

What are the styles of documenting? (strict, follow conventions or freedom?)

What tools do you use (if any)?

- For developing environment (e.g., IDEs/editors/compliers)?
- For testing?
- For team cooperation (e.g. source-safe)?

How strongly are the selected tools reflected on the software development processes?

b. Do you have any difficulties in doing Agile?

Is there any circumstance that you do not get the full management support? (e.g., for resources you need)

What is the level of your staff/co-worker competency and communication?

Do your clients (stakeholders, business-side) always involve and cooperate?

Is there always high level of trust among all participants in a project?

c. In your experience, how would you evaluate the performance of being more mature with agile?

Do you achieve more client satisfaction?

Do your software products have less defects/errors?

Does the development process have shorter time? Or in other words, do your clients have shorter time to market?

How much costs reduction (in percentage, if any) does each project have?

Does your company still maintain the standard compliance?

d. What do you want to do differently/ improve?
    - With respect to the software development processes?
    - With respect to the practices?
    - With respect to the supporting tools? Are they replaceable?

VI. Closure

a. Are there any related questions that I forgot to ask?
b. Is there anything else would you like to tell me?
c. Can I cover the unclear issues after this interview via emails or can we have a meeting again?