UNIVERSITY OF VAASA

THE SCHOOL OF TECHNOLOGY AND INNOVATION

SOFTWARE ENGINEERING

Laura Mustonen

TESTING AND IMPROVING A CONTINUOUS REQUIREMENTS RISK PROFILING METHOD

A Case Study on Agile Software Projects

Master's thesis for the degree of Master of Science in Technology submitted for inspection, Helsinki, 30th of March 2018.

Supervisor Jouni Lampinen

Instructor Tero Vartiainen

TABLE OF CONTENTS

1 INTRODUCTION 9
1.1 Background of the study 9
1.2 Objectives of the study 10
1.3 Key concepts and limitations of the study 11
1.4 Structure of the study 12

2 AGILE SOFTWARE DEVELOPMENT 13
2.1 Adoption of agile software development models 13
2.2 Common approaches to agile software development 17
2.2.1 Extreme Programming 18
2.2.2 Scrum 20
2.2.3 Lean software development and Kanban 22
2.3 Common practices related with agile software development 25
2.3.1 Continuous Integration 26
2.3.2 Continuous Delivery 27
2.3.3 DevOps 28

3 REQUIREMENTS ENGINEERING IN SOFTWARE DEVELOPMENT 31
3.1 The purpose of software requirements 31
3.2 Requirements process in software engineering 32
3.3 Approaches and techniques for requirements engineering 34
3.3.1 Discovering requirements 35
3.3.2 Experimenting with requirements 35
3.3.3 Prioritizing requirements 35
3.3.4 Specifying requirements 36
3.4 Common challenges of requirements engineering and management 36

4 MANAGING RISK IN SOFTWARE PROJECTS 38
4.1 What is software project risk and risk management? 39
4.2 Six dimensions of software project risk 40
4.3 Focus on software project requirements risk 43
4.3.1 Identifying and categorizing the requirements risk 44
4.3.2 Resolving requirements risk 46
4.4 Risk management methods and tools 47
4.5 Introducing the Continuous Requirements Risk Profiling Method 48

5 RESEARCH PROCESS AND METHODS 55

5.1 Interpretive case studies 55
5.2 Theme-centered interview as a research method 56
5.3 Qualitative data analysis based on themes 58
5.4 Validity and reliability of the study 60
5.5 Designing the study 61
5.5.1 Description of the case company 61
5.5.2 Purpose of the interviews 62
5.5.3 Applied assessment criteria 62
5.5.4 Planning and validation of the data collection method 64
5.5.5 Structure of the interviews 65

6 TESTING AND IMPROVING THE CONTINUOUS REQUIREMENTS RISK PROFILING METHOD 67
6.1 Conducting the theme-centered interviews 67
6.2 Thematic analysis for interview data 69
6.2.1 Coding applied to dataset and identified themes 70
6.2.2 Thematic map of interview data 71
6.3 Analyzing the method feasibility in agile software development projects 72
6.3.1 Theme 1: Managing requirements risk in agile software projects 73
6.3.2 Theme 2: Experiences on requirements risk in agile software projects 76
6.3.3 Theme 3: Assessment of the method completeness, accuracy and understandability 80
6.3.4 Theme 4: Assessment of the method usefulness and feasibility 84
6.4 Proposing improved Continuous Requirements Risk Profiling method 87
6.4.1 Tailoring method instructions to case company environment 87
6.4.2 Additions and changes to checklist risk items 88
6.4.3 Improvements to risk profiling table 92
6.4.4 Introducing the tailored requirements technique toolbox 94

7 DISCUSSION 96

8 CONCLUSIONS 99

REFERENCES 101

APPENDIX A 105

APPENDIX B 109

TERMS AND ABBREVIATIONS

Agile	Software development approach
CI	Continuous Integration
CD	Continuous Delivery
DevOps	Development and Operations
Kanban	“Signboard”, task visualization tool
Lean	Organizational strategy
MIS	Management Information Systems
Scrum	Product development process framework
TDD	Test-Driven Development
XP	Extreme Programming

LIST OF TABLES

Table 1 Summary of Twelve Principles of Agile Software introduced by Beck et al. (2001) as part of The Agile Manifesto. 14

Table 2 Comparison of four characteristics in traditional and agile methods by Cao & Ramesh (2007: 41–42). 16

Table 3 Summary of XP Practices presented by Beck (1999: 71). 19

Table 4 Seven lean principles applicable to software development presented by Poppendieck & Cusumano (2012: 28–30). 23

Table 5 Six main benefits of adopting Continuous Delivery in large publishing company observed by Chen (2015: 52–53). 28

Table 6 Requirements activities presented by Hickey & Davis (2004: 67). 33

Table 7 The four qualities usually causing software engineering problems (Brooks 1986: 11–12). 37

Table 8 Summarizing six dimensions of software project risk presented by Wallace et al. (2004: 117). 41

Table 9 Summary of requirements risk categories introduced by Mathiassen et al. (2007) and later complemented by Tuunanen et al. (2015: 4027). 44

Table 10 Requirements development technique types (Mathiassen et al. 2007: 576). 46

Table 11 Summary of four types of risk management models and their characteristics identified and synthesized by Iversen et al. (2004) and later analyzed by Mathiassen et al. (2004: 35–36). 47

Table 12 Six software project key risk drivers presented by Tiwana & Keil (2004: 75). 48

Table 13 The initial risk resolution pattern presented by Tuunanen et al. (2016). 50

Table 14 The initial requirements phase checklist by Tuunanen et al. (2015). 51

Table 15 The initial design phase checklist by Tuunanen et al. (2015). 51

Table 16 The initial implementation phase checklist by Tuunanen et al. (2015). 52

Table 17 The initial risk-profiling table by Tuunanen et al. (2015). 53

Table 18 Six steps of thematic analysis presented by Braun & Clarke (2006: 87). 59

Table 19 Assessment criteria adapted from MIS Success Measures presented by DeLone & McLean (1992: 84–85). 63

Table 20 . The planned structure for the theme-centered interviews. 65

Table 21 Summary of interviewed professionals and their background information. 68

Table 22 Coding applied to dataset and identified themes. 70

Table 23 Conducting risk analysis using the Continuous Requirements Risk Profiling method and applying the risk resolution pattern adapted from Tuunanen et al. (2016) tailored to case company context. 88

Table 24 Additions suggested to requirements risk checklists. 89

Table 25 Requirements phase checklist initially presented by Tuunanen et al. (2016) and improved by feedback collected in this study, additions and changes **bolded**. 90

Table 26 Design phase checklist initially presented by Tuunanen et al. (2016) and improved by feedback collected in this study, additions and changes **bolded**. 90

Table 27 Implementation phase checklist initially presented by Tuunanen et al. (2016) and improved by feedback collected in this study, additions and changes **bolded**. 91

Table 28 The changes suggested to presented relative impacts of risk items. 92

Table 29 Indicative risk impact levels initially presented by Tuunanen et al. (2016) and improved by feedback collected in this study, additions and changes **bolded**. 93

Table 30 Requirements risk resolution technique list and categorization, adapted from Mathiassen et al. (2007: 594–596) and filtered based on the results to specify the most well-known techniques among the interviewed professionals. 94

Table 31 The initial requirements risk resolution techniques list and categorization adapted from Mathiassen et al. (2007: 594–596) by Tuunanen et al. (2016). 109

LIST OF FIGURES

Figure 1 Visualization of the differences between waterfall, iterative and Extreme Programming approaches by Beck (1999: 70). 18

Figure 2 Generic DevOps production and delivery process presented by Ebert et al. (2016: 95). 30

Figure 3 The relationships between presented project characteristics and different types of risk revealed by Wallace et al. (2004: 121). 42

Figure 4 Applying continuous requirements risk profiling and management in continuous development (Tuunanen et al. 2015: 4020). 49

Figure 5 Theme interview process presented by Hirsjärvi & Hurme (2000: 67). 58

Figure 6 The final thematic map illustrating conducted thematic analysis on interview data, reviewed for correspondence and consistency of themes. 72

VAASAN YLIOPISTO
Teknillinen tiedekunta
Tekijä: Laura Mustonen
Diplomityön nimi: Testing Feasibility of a Continuous Requirements Risk Profiling Method – A Case Study on Agile Software Projects
Valvojan nimi: Jouni Lampinen
Ohjaajan nimi: Tero Vartiainen
Tutkinto: Tieto- ja tietoliikennetekniikan koulutusohjelma
Pääaine: Ohjelmistotekniikka
Opintojen aloitusvuosi: 2012
Tutkielman valmistumisvuosi: 2018 **Sivumäärä:** 114

TIIVISTELMÄ:

Onnistunut vaatimusmäärittely on yksi avaintekijä koko ohjelmistoprojektin menestykselle, ja näin myös vaatimuksiin liittyvien riskien tunnistaminen ja hallitseminen ovat tärkeä osa laadukkaiden ohjelmistojen tuottamista. Tästä huolimatta vaatimusmäärittelyn riskienhallintaan kehitettyjä malleja on vähän, eikä kirjallisuudesta juurikaan löydy tuloksia mallien hyödynnettävyydestä ohjelmistoteollisuudessa. Tässä diplomityössä testataan yhden vaatimusmäärittelyn riskienhallintaan kehitetyn menetelmän soveltuvuutta ketterissä ohjelmistoprojekteissa, sekä ehdotetaan parannettua versiota mallista. Testattava menetelmä käsittää riskien tunnistamisen, priorisoinnin ja riskeihin puuttumisen erilaisten ratkaisumallien avulla. Tutkielman tavoitteena on saada tietoa kokevatko ketterissä ohjelmistoprojekteissa työskentelevät ammattilaiset menetelmän käyttökelpoiseksi ja hyödylliseksi, sekä kuinka mallia tulisi parantaa, jotta se voitaisiin ottaa käyttöön case-yrityksessä.

Työ toteutettiin tulkitsevana tapaustutkimuksena, jossa samasta yrityksestä tarkasteltiin useampaa tutkimuksen kohderyhmään sopivaa projektia. Tutkimuksen tiedonkeruumenetelmänä toimivat puolistrukturoidut teemahaastattelut, joissa menetelmää arvioitiin toteuttamalla kullekin käsitellylle projektille mallin mukainen riskianalyysi. Haastatelluilta asiantuntijoilta edellytettiin osallistumista ketteriä ohjelmistokehitysmenetelmiä käyttävän projektin vaatimusmäärittelyntyöhön. Haastatteluilla kerätty kvalitatiivinen data analysoitiin käyttäen temaattista analyysia.

Tutkimuksen tuloksista havaittiin, että malli auttoi asiantuntijoita tunnistamaan eri tyyppisiä vaatimusmäärittelyyn liittyviä riskejä ja priorisoimaan niitä yleisellä tasolla. Asiantuntijat kokivat mallin käytön hyödylliseksi ja sopivan niihin ketteriin ohjelmistoprojekteihin joissa he työskentelivät. Mallin tarjoamia riskien ratkaisuehdotuksia tulisi kuitenkin kehittää edelleen. Keskeisimpinä suosituksina tutkimukseen on mallin kehittäminen ratkaisuehdotuksien osalta, sekä käytäntöön vaatimusriskeinhallintaan liittyvän tiedon edelleen kerääminen ja jakaminen vastaavien työkalujen kautta.

AVAINSANAT: Vaatimusmäärittely, Ketterä ohjelmistokehitys, Ohjelmistoprojektin riskienhallinta

UNIVERSITY OF VAASA
Faculty of technology
Author: Laura Mustonen
Topic of the Thesis: Testing Feasibility of a Continuous Requirements Risk Profiling Method – A Case Study on Agile Software Projects
Supervisor: Jouni Lampinen
Instructor: Tero Vartiainen
Degree: Degree Programme in Software Engineering
Major: Software Technology
Year of Entering the University: 2012
Year of Completing the Thesis: 2018 **Pages:** 114

ABSTRACT:

As requirements play key role in the success of a software development project, identifying and mitigating requirements related risks becomes an important factor in improving software quality. Still, only few methods are offered for that purpose and little results of the feasibility of such methods in industry are reported. In this thesis, feasibility of one requirements risk management methodology was tested in agile software projects and an improved version of the method proposed. The tested method consists of identifying, prioritizing and resolving risks using predefined checklists, patterns and techniques. The objectives of the study were to gain knowledge do professionals working in agile software projects find the method feasible, are such methods found useful and how the method should be improved so that it could be taken into use in the case company.

The study was conducted as an interpretive case study which covered several agile software projects from the case company. The primary data collection method for the study were semi-structured theme-centered interviews, in which the method was tested and evaluated by conducting a requirements risk analysis for each of the case projects. The key selection criteria for the interviewees was participation to requirements work and use of some agile software development methodology. The collected qualitative interview data was analyzed using thematic analysis.

Based on the results of this study, it was observed that the tested method helped professionals to identify different type of requirements risks and to prioritize those on high level. The interviewed professionals found the tested method useful and feasible in the agile software projects they were currently working with. However, it was also observed that the resolution proposals provided by the method would still need further development. For researchers, the study provided empirical evidence on the feasibility of the method and several suggestions for further research. For professionals working in industry, the study provided one empirically validated method for managing requirements risk, and encouragement for collecting the existing requirements risk management knowledge and sharing it with corresponding methods and tools.

KEYWORDS: Requirements engineering, Agile software development, Risk management

1 INTRODUCTION

This study is a Master's thesis for a degree in Software Engineering, and examines the topics and theoretical concepts related software requirements risk management in agile software projects. In the study, we seek to provide new information and aspects to the research topic by testing feasibility of one recently proposed requirements risk management framework, referred here as the Continuous Requirements Risk Profiling method. As the requirements risk management framework has not yet been adopted by industry professionals, it still relies mainly on theoretical knowledge and lacks empirical validation by its targeted user group: industry professionals working with requirements engineering and management in agile software projects. The goal of this study is to fill this gap by testing and using the Continuous Requirements Risk Profiling method in case company with a group of experienced industry professionals. In this chapter, following is discussed in more detail: the background and motivation of the study and choosing research topic, the objectives for testing the method, the key concepts and limitations of the study and structure of the rest of this report.

1.1 Background of the study

Theoretical background of the research topic is related to previous research done by Mathiassen, Saarinen, Tuunanen & Rossi in *A Contingency Model for Requirements Development* (2007), which introduced a framework for identifying, categorizing and mitigating requirements risk. Tuunanen, Vartiainen, Ebrahim & Liang (2015) later presented the Continuous Requirements Risk Profiling Method basing on the theoretical knowledge by Mathiassen et al. (2007) and transforming it into a risk management method corresponding the needs of industry professionals working in agile software projects. Continuous Requirements Risk Profiling Method presents tools for continuous risk identification, prioritization and resolution through the software development lifecycle. The method targets specially to meet the needs of software projects using agile methodologies or employing DevOps practices, but is agnostic for the chosen development methodology (Tuunanen et al. 2015). As there are not any reported results about the method use this far, this study

targets to provide such by reviewing, testing and validating the method in case company context.

Motivation for the study topic comes from both previous research and practice. In previous research, Tuunanen et al. (2015) have pointed the absence of a model for requirements risk profiling and prioritization, which would be feasible in iterative and agile software development projects. From industry professionals' point of view, the lack of such tools and methods also had been acknowledged in the case company of this study. The professionals working with requirements management and risk management in case company pointed out that there were no such tools available and sometimes it was difficult to prioritize the requirements related risks. Using agile development approaches and practices such as DevOps, Continuous Integration or Continuous Delivery models for software projects is also increasing trend and for example Cao & Ramesh (2007) present that the group of agile methods have already gained a lot of attention. These agile methods rely on continuous and iterative workflows, which sets its own requirements to the techniques used in requirements management (Tuunanen et al. 2015).

1.2 Objectives of the study

The main objective of the study is to test the Continuous Requirements Risk Profiling method introduced by Tuunanen et al. (2015) in software projects using agile development practices and collect feedback and insights from industry about the feasibility and usefulness of the method. Based on the collected empirical evidence, also a validated and improved version of the method will be proposed to be used in the case company. By presenting this one empirically validated view of the method, we hope to help both professionals working in industry to manage requirement risk in agile software projects and researchers in developing the model and finding future research topics on this area. The main research questions of this study are:

1. Does the developed continuous requirements risk profiling method fit the needs of agile software development projects?

2. Would the developed continuous requirements risk profiling method help industry professionals to identify such project characteristics that are seen as risks for the project success?
3. How the continuous requirements risk profiling method should be improved, so that industry professionals would find it easy to use and useful in their everyday work?

In this study, we try to seek answers to the research questions by testing the Continuous Requirements Risk Profiling method with several agile software development projects in case company context. We do this by conducting requirements risk management interview sessions with professionals, where we assess a risk profile for one specific project and evaluate the method. In later parts of the study, we then analyze the interview results with qualitative methods, such as thematic analysis, and evaluate the results.

1.3 Key concepts and limitations of the study

Key concepts of this study are *requirements engineering and management*, *agile software development*, and *risk management*.

As the Continuous Requirements Risk Profiling method is developed specially to meet the needs of agile software projects and projects employing DevOps practices, the case projects examined in this study are limited to those. Conducting the study in case company context also sets some limitations to the repeatability of the study and generalizability of the results. Factors possibly affecting to the results are for example the business domain and environment the case company is operating in, and the type of the software solutions that are developed in examined projects. Even though the results will represent several different types of projects that fit defined characteristics, the results might be hard to generalize to apply all agile software projects.

1.4 Structure of the study

The rest of the study consists of seven main chapters, which are structured as follows. This first chapter introduced the background, objectives, key concepts, limitations and overall structure of the study. The second chapter introduces the reader to agile software development methodologies and some widely adopted agile practices, to give better understanding of the context of this study. The third chapter is focused on requirements engineering in software development, and gives the reader basic understanding of the purpose of software requirements, the requirements process and activities, and some common ways of doing requirements engineering. The fourth chapter discusses about the overall concepts of risk management, giving later more focused view on requirement risk management. In addition, the Continuous Requirements Prioritization method, tested in the empirical part of the study, is introduced in the end of fourth chapter. The fifth chapter describes the design of the study, including the research process and methods used for testing and improving the Continuous Requirements Prioritization Method. The sixth chapter describes how the study was conducted and the results analyzed. The seventh chapter presents the conclusions made based on the empirical results of the study.

2 AGILE SOFTWARE DEVELOPMENT

This chapter introduces the past and present of agile in software development. We discuss why and when the adoption of agile software development methods happened in industry, introduce some widely adopted approaches to agile software development to give better understanding of the environment and context of this study. In addition, software engineering concepts such as DevOps, Continuous Integration and Continuous Delivery are discussed, as those are common technical principles often related to practicing agile software development methods.

The agile methods seek to respond the challenges of the ever-changing environments, and share common characteristics like short development iterations, frequent system releases, on-site customer participation, use of peer review techniques and simple and incremental designs (Cao & Ramesh 2007: 41–42). The recent research shows that agile software development methods and principles have become the preferred choice in variety of software organizations, despite the organizational size. Even though many of the organizations involved in agile software development do not promote to directly follow certain agile methodology, researchers have found that many of the principles base on popular methodologies such as Scrum and Extreme Programming. (Ramesh, Cao & Baskerville 2010: 449–450, 454, 474.)

2.1 Adoption of agile software development models

The wider acknowledging of agile software development dates back to change of 90's and 00's. *The Agile Manifesto* (Beck, Beedle, Bennekum, Cockburn, Cunningham, Fowler, Grenning, Highsmith, Hunt, Jeffries, Kern, Marick, Martin, Mellor, Schwaber, Sutherland & Thomas), often referred as declaration of agile software development philosophy and values, was published in 2001 by a group of software professionals. This group of software professionals had been looking for alternative ways to do software to then popular approaches and for example, Beck (1999) had already presented an agile approach called *Extreme Programming* before the actual manifesto was published. The

manifesto states, that when searching for better ways to do software compared to traditional strictly planned approach, the authors have come to value following:

> "Individuals and interactions over processes and tools.
> Working software over comprehensive documentation.
> Customer collaboration over contract negotiation.
> Responding to change over following a plan." (Beck et al. 2001.)

The manifesto is supplemented with *The Twelve Principles of Agile Software*, which describes the working principles that had led to the values presented in manifesto. The principles also highlight embracing change, delivering early and frequently, the importance of open and effective communication both with clients and inside development team and focusing on the essential work, leaving invaluable work undone (Beck et al. 2001). These twelve principles by Beck et al. (2001) are summarized in table 1.

Table 1 Summary of Twelve Principles of Agile Software introduced by Beck et al. (2001) as part of The Agile Manifesto.

1.	Change is customer's competitive advantage. Changes to requirements should be accepted even after the system is put to development.
2.	Providing working deliveries of software frequently, preferring short intervals between deliveries.
3.	Collaboration between the business and development must be daily and work throughout the project.
4.	Most important part of the project are motivated individuals, and providing them the circumstances to get the work done.
5.	Preferring face-to-face conversations for communication over formal meetings.
6.	Progress is measured in terms of working software.
7.	Development should be made on pace that is sustainable also on long-term.
8.	Paying attention on good design and technical excellence improve agility.
9.	The highest priority in agile development is satisfying the customer by delivering valuable software early and continuously.
10.	Focusing only on essential work.
11.	Self-organizing teams create the best solutions.
12.	The team targets to continuous improvement and reflects its way of working regularly.

As *Agile Manifesto* and *Twelve Principles of Agile Software* presented, accepting and responding to change is a fundamental part of agile. For gaining some deeper understanding about agile, it is also important to comprehend where the need for agile methodologies has originated, and where such principles are likely to be effective. Some researchers have searched explanations from organizational theory and examining the environments where agile methods are usually applied. Cao & Ramesh (2007) present that rapidly changing environment, often observed in a software project through highly volatile requirements, is one of the most important reasons behind introducing agile software development approaches. (Cao & Ramesh 2007: 41–42, 47.)

Cao & Ramesh characterize rapidly changing environments with tight schedules and evolving requirements that can become obsolete even before project completion (2007: 41). The effect of this kind of environments and need to response to change should not come as a surprise, as already Brooks (1986) argued that changeability is one of the core qualities of software. Brooks (1986) stated that this change can be seen also as an inevitable and inherent part of software, as well as the complex environments the software systems are operated. As in some cases the software might be the only part that can be mold at any cost, it is the one to be molded. (Brooks 1986: 10–12.)

When examining the general differences between traditional methods, such as waterfall, and agile methods, these could be explained based on four characteristics: environment, values, beliefs and implementation of practices (Cao & Ramesh 2007: 41–42). These characteristics are explained more detailed in table 2.

Table 2 Comparison of four characteristics in traditional and agile methods by Cao & Ramesh (2007: 41–42).

Environment	Traditional methods fit to more stable environments where quality is the major concern, as agile methods excel in dynamic environments where requirements and technology are volatile and time to market is critical.
Values	Traditional methods trust on planning, control, predictability and high assurance, as agile methods promote collaboration, interaction and adaptability based on "*Manifesto for Agile Software Development*" (Beck et al. 2001).
Beliefs	Traditional methods believe in having a complete and accurate specification and controlling change, as agile methods believe that requirements are created throughout the development and change is unavoidable.
Implementation of practices	Traditional and agile often implement the same software development practices in different ways. For example, releases are much more frequent in agile methods compared to traditional methods.

Despite the popularity, the agile methods have also been criticized as a set of ad-hoc practices that have been created to solve the practical problems occurring in rapidly changing environments. Thus, Cao & Ramesh (2007) examined whether the practices of agile software development methods had correspondence to the research streams in organizational theory applicable to the context. They present that even though agile development methods might seem to be evolved from a set of common best practices as an answer to this kind of challenging environments, there is also consistency between the organizational theories and agile methods. Correspondence was found in *"Dynamic Capabilities theory"* which helps to explain why to use agility, *"Coordination theory"* which explains the call for transformed coordination mechanisms in agile environments and *"Double-Loop Learning theory"* which reasons the emphasis on continuous learning in agile approaches. (Cao & Ramesh 2007: 41, 46–47). This gives also theoretical verification for the validity of otherwise practitioner-led adoption of agile principles and methodologies.

What would make the results presented by Cao & Ramesh (2007) interesting for someone who is implementing agile software development in practice is that the found correspondence could help to understand the validity and applicability of the agile development methodologies. Thus, it is suggested that rather than focusing only to the question if agile methods have anything actually new in them, it would be more valuable to understand the conditions under which certain agile practices could be applicable and effective. (Cao & Ramesh 2007: 46–47). Therefore, as there is many different kinds of agile methodologies and practices to choose from, the choice of certain methodology should be justifiable. In next chapter, some common approaches to agile software development are presented.

2.2 Common approaches to agile software development

From traditional software development methodologies point of view, once the implementation has been started, change in requirements is seen as expensive and unwanted (Beck 1999: 70). As a reflection to agile thinking and the willingness to take advantage of the changes affecting to software development process, several different approaches and methodologies to agile software development have emerged. In this subchapter, some of the most widely adopted methodologies are discussed in more detail, including Extreme Programming (abbreviated as XP), Scrum, Lean software development and Kanban. Of the discussed agile software development methodologies, Scrum is the most important for understanding the environment where the study is conducted.

Extreme Programming (XP) was the first one of the agile approaches to become popular, and was more focused on introducing technical practices that help to improve agility, such as Test-Driven Development (TDD) and Continuous Integration (CI). Scrum is a software development approach that replaces the traditional project management with development iterations of two to four weeks. Despite the popularity, Scrum mainly project management centered and does not claim to be a full methodology including all the technical practices needed in agile software development. Both Scrum and XP rely on defining certain rules and roles that guide the software development process. Kanban has roots in manufacturing industry and Lean ideology, but of the presented three approaches, Kanban

is the most recently adopted one in software development. Lean is a more organization-wide approach for optimizing and improving value streams and workflows, as Kanban is a visualization tool for previous. (Poppendieck & Cusumano 2012: 30–31.)

2.2.1 Extreme Programming

One of the agile development approaches willing to welcome change to the software development process is Extreme Programming (XP). Beck describes in his article *"Embracing change with Extreme Programming"* (1999) that XP turns the regular software process sideways, doing analysis, design, implementation, and testing little by little throughout the development. (Beck 1999: 70–71). Visualized in Figure 1, the XP way is compared to the traditional waterfall approach to perform software development activities in one sequence, and to iterative approach where software development activities are divided into several iterations.

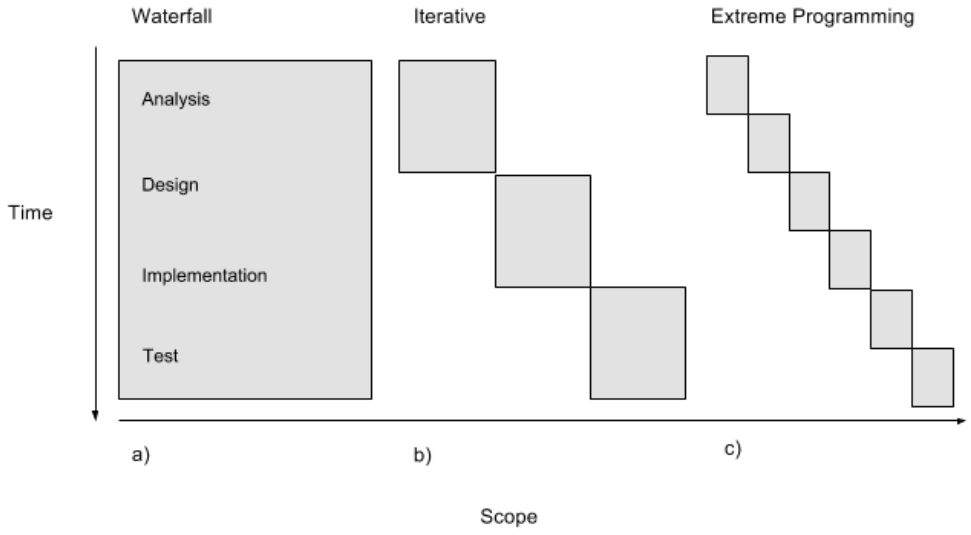

Figure 1 Visualization of the differences between waterfall, iterative and Extreme Programming approaches by Beck (1999: 70).

In XP system requirements are referred as stories, which summarize the overall analysis of the system. Each of the created stories should be business-oriented, estimable and testable. For each system release the customer may select stories which he finds the most valuable, being also informed about the cost of each story and the team's measured implementation speed. XP approaches the requirements analysis from perspective that you can never explore all of the requirements, if you never implement any. Thus, the first analysis phase is recommended to be kept as compact as possible. (Beck 1999: 71–72.)

Beck (1999) summarizes the XP ideology into thirteen major XP practices including Planning game, Small releases, Metaphor, Simple design, Test, Refactoring, Pair programming, Continuous integration, Collective ownership, On-site customer, 40-hour week, Open workspace and Just rules. These practices are presented in more detail in table 3.

Table 3 Summary of XP Practices presented by Beck (1999: 71).

Planning game	Programmers provide implementation cost estimates for stories, based on which the customer chooses scopes and timings for each release.
Small releases	Releases are made often and the system is put into production before solving the full problem.
Metaphor	Metaphors are used in communicating the shape of the system between customer and programmers.
Simple design	At all times, the design is kept in its simplest form that solves the problem, and does not contain duplicity.
Tests	Programmers are responsible of creating unit tests for the code. Customer is responsible of the functional tests for the stories implemented in each iteration.
Refactoring	The design of the system is refactored over time, and each improvement must keep all previously implemented tests running.
Pair programming	All production code is a result of pair programming, which means two programmers working on same workstation.

Continuous integration	All new code is integrated to the existing system as soon as possible, and changes are not accepted unless the build and tests pass.
Collective ownership	No one own certain code, any programmer should improve any part of the system when seeing a chance for it.
On-site customer	Customer has a full-time representative working with the team.
40-hour week	No one is allowed to work overtime on two following weeks, and overtime is seen as a sign of deeper problems that need to be solved.
Open workspace	Team works in one open room, where the pair programmers work in the center.
Just rules	Every team member must accept the rules, but the team can also change the rules whenever they feel it is required.

2.2.2 Scrum

Scrum is product development framework, defined in "*The Definitive Guide to Scrum: The Rules of the Game"* developed and sustained by Schwaber & Sutherland (2016). It has gained a lot of attention in software industry since 2001 and has become a popular alternative for traditional project management (Poppendieck & Cusumano 2012: 30). The Scrum framework includes definitions for values, team roles, events, artifacts and language, such as definition of "Done" that should be used and agreed when practicing Scrum. The Scrum framework has been used in product development since the early 1990s. The ideology of Scrum has foundations on empirical process control theory, which builds on *transparency*, *inspection* and *adaption*. To add risk control and predictability, Scrum uses an iterative, incremental development approach. (Schwaber & Sutherland 2016.)

The self-organizing and cross-functional Scrum teams consists of people with roles of Product Owner, Scrum Master and development team members. The product owner is responsible of the value delivered by the developed product, and managed the product

requirements that are put into the form of a product backlog. Scrum defines that product owner must a one person, and the whole organization must respect the decisions made the product owner. The scrum master is servant-leader for the whole Scrum team, and works to help the team to maximize the created value. Scrum master helps the team to work agile, finds ways to remove possible obstacles and facilitates the Scrum events. The development team is the group of professionals who work on the increment developed in each iteration. The development team organizes and manages their own work, and should optimally consist between three to nine team members. (Schwaber & Sutherland 2016.)

The defined Scrum events are for purposes of inspection and adaption, and target to minimize the need for any additional meetings. These events are Sprint, Sprint planning, Daily Scrum, Sprint Review and Sprint Retrospective, which each have a defined maximum duration. Most important is sprint, which stands for the maximum one-month length iteration when a new product increment is developed. Within a sprint no changes a done to the sprint or quality goals, which also means that the requirements should not be changed. Doing the work on short-enough sprints aims to limit the risk related to each increment to the length of the sprint. In extreme cases, if environment changes so much that the sprint goal and requirements do not make sense anymore, the sprint may be cancelled. Besides the development work, each sprint contains the Sprint planning, Daily Scrum, Sprint Review and Sprint Retrospective events. (Schwaber & Sutherland 2016.)

Sprint planning is for planning and agreeing the work that will be done in the upcoming iteration. The input comes from product backlog items, to which the development team gives work estimates. Daily scrum is a brief daily meeting, where the development team discusses the work that was done on previous day, what they are going to do next and have any problems arose. Sprint review is an informal meeting, where the developed increment is presented to the whole scrum team and stakeholders to collect feedback. Sprint retrospective is meeting at the end of each sprint where scrum team review their ways of working, and plan actions how to improve. (Schwaber & Sutherland 2016.)

The scrum artifacts Product Backlog and Sprint Backlog target to maximize the transparency of key information. Product backlog contains all product requirements, created, described and ordered by the product owner. It is never complete, and contains all possible requirements, functionalities, features, changes or fixes needed to the product in future releases. Changes for example in business requirements, technology or environment can cause changes to product backlog items. Sprint backlog defines the subset of product backlog items selected to specific sprint and the plan how an increment that implements selected items can be delivered. Increment is the useable and deliverable version of product completed at the end of each sprint. (Schwaber & Sutherland 2016.)

The rules defined in Scrum methodology differ from rules in XP for example on their strictness: The Scrum framework is immutable. Even though in XP everyone must follow certain rules, the team can change the rules as Beck (1999) stated in the "Just rules" principle. According to Schwaber & Sutherland (2016), as long as you want to call something Scrum, you may not implement the methodology only partially or change the rules. However, they state than Scrum can be used as a container for other software development practices. Being capable to only partially implement Scrum is a common pitfall, and among industry professionals that kind of incomplete implementations are referred as "Scrum-but" (Rinko-Gay 2013).

2.2.3 Lean software development and Kanban

Ebert, Abrahamsson & Oza (2012) analyze the roots and use of lean in software engineering in article "*Lean Software Development*". Lean development could be summarized as a product development paradigm, first introduced in manufacturing, which has a comprehensive focus to create value for the customer, eliminate waste, optimize the value streams, empower people, and continuous improvement. These five elements also build the lean product development cycle, which is an iterative cycle consisting of previous activities. Ebert et al. present that the idea of lean development is in line with an old wisdom in software development, according to which most features do not add value, but instead add unnecessary cost and complexity. (Ebert et al. 2012: 21–23.)

In past decade, the adoption of agile and lean principles to make also software development more efficient has been increasing. In software industry, the move towards lean started with agile programming methods. (Ebert et al. 2012). We can notice previous also from the previously introductions to Scrum and XP, often the agile principles are more focused on software teams or software projects than enterprise-level. Ebert et al. (2012) argue that such focus can lead to short-term improvements, but might result as a negative impact to the overall software life-cycle costs. They present that in the beginning lean software development was much connected to agile, but lately more diversity has been introduced. Even though all lean principles such as mathematical production models do not fit to software development, there is still a lot the software development organizations can learn from. (Ebert et al. 2012: 22–24.)

To apply lean manufacturing management principles to product development and software engineering, Poppendieck & Cusumano (2012) present that *lean* should be viewed as a set of principles rather than practices. They argue that this kind of approach makes the ideology more applicable to different environments and use cases, and can lead to process and quality improvements like when applied in manufacturing. Following previous ideology, they also describe seven principles that can be used to get started with lean software development. These are to optimize the whole, eliminate waste, build quality in, learn constantly, deliver fast, engage everyone and keep getting better. (Poppendieck & Cusumano 2012: 28). Previous principles are explained more detailed in table 4.

Table 4 Seven lean principles applicable to software development presented by Poppendieck & Cusumano (2012: 28–30).

Optimize the whole	Deep understanding of the customer's needs and values, and how those can be solved with software are the foundation of lean software development. Furthermore, the value of software does not only come in the implementation, but also design and deployment are necessary for achieving the value.
Eliminate waste	Lean treats as waste anything that does not add customer value or increase knowledge on how the value can be delivered more

	effectively. In software development, these are for example unneeded features, lost knowledge, handovers, half-finished work, multitasking and time spent debugging and fixing defects.
Build quality in	As the waste has often its roots in large amounts of half-finished work, boundaries between functions and the loss of knowledge and time caused by previous boundaries. Thus, previous should be avoided and continuously integrating small units into larger software systems favored.
Learn constantly	Development is about creating knowledge and embodying the created knowledge into a product. In lean, this can be approached in two ways: either by delaying the most expensive-to-change decision as long as possible, or by delivering first version fast and learning from customer feedback.
Deliver fast	Lean environments favor frequent production releases, often occurring weekly, daily or continuously. Releasing software frequently moves the idea of software development from a project towards a flow system.
Engage everyone	When software development is viewed a flow process, the organizational structure should be reminiscent of line business units containing also supporting functions, instead of one separate IT-department.
Keep getting better	According to lean thinking, every process should be improved continuously. Thus, adoption of popular agile practices should be viewed as a starting point and those practices improved continuously to fit best the problem in hand.

Poppendieck and Cusumano (2012) argue that they see same kind of orientation also in other popular agile software development methodologies, such as Extreme Programming (XP) and Scrum. The emphasis on previous methodologies and lean is on reducing wasted time and labor, focusing on creating value to customer. However, they point out that the main difference between lean and agile software development methodologies lies in the fact that lean is a more complete and organization wide approach than for example XP or Scrum. As lean is a more product oriented approach, Poppendieck & Cusumano (2012)

argue that it also provides better support integrating for example user experience and product design teams, or other supporting functions to software development. While agile methodologies apply only to software teams, lean gives organization wide guidance for choosing development practices appropriate to individual contexts and situations. (Poppendieck & Cusumano 2012: 27, 30–32.)

Kanban is one tool for presenting workflows, tasks and value streams in lean. In agile software development, Kanban "signboard" is often used to visualize software development and production operations. Ahmad, Markkula & Ovio (2013) point out that at least from the research literature point of view, the Kanban approach is one of the most recently adapted methods in agile software development. In recent years, the popularity of Kanban as a part of agile software development practices has increased and Ahmad et al. (2013) describe that the movement is mainly practitioner led. Use of Kanban can help the teams to for example limit the work in progress according to their capacity, visualize problems in development process and to maintain a steady flow of tasks. (Ahmad, Markkula & Ovio 2013: 9–10.)

2.3 Common practices related with agile software development

Besides previous project-oriented agile software methodologies, providing guidance for the overall workflow, also some technical practices have become popular in agile software development. Here we present some common technical practices that are closely related to the topic of this study and important for understanding the environment where the study was conducted. These practices are Continuous Integration (CI), Continuous Delivery (CD) and DevOps. Projects employing DevOps practices were another target group in this study for testing the Continuous Requirements Risk Profiling method, but no projects fully implementing the DevOps practices were found from the case company.

Continuous Integration (CI) is an agile software development practice, where the software development team integrates their code frequently. CI usually employs automated builds

and automated tests for verifying the changes made to code base. (Fowler 2006). Continuous Delivery (CD) takes CI still a step further, and covers the delivery of software too. CD extends the automation to so that a new system could be delivered to production after every change in the code base. (Chen 2015: 50). DevOps targets to a shift that integrates the development, delivery and operations in software development organizations. DevOps employs a high degree of automation to all phases of software development process to reach its goal. (Ebert, Gallardo, Hernantes & Serrano 2016).

2.3.1 Continuous Integration

According to Fowler (2006) Continuous Integration (often referred with abbreviation CI) is "a software development practice where members of a team integrate their work frequently, usually at least daily". Originating from one of twelve Extreme Programming practices, Continuous Integration aims to change the traditional perception of integrating software as a long and unpredictable process. Continuous Integration relies on extensive automation, including automated builds with automated tests verifying each integration. This type of verification targets to detecting errors or conflicts caused by the code changes as quickly as possible, avoiding later major rework. (Fowler 2006.)

Fowler (2006) explains that at its simplest, Continuous Integration practice does not require any particular tooling, but use of Continuous Integration server is often found useful. Continuous Integration is more implementation related practice, which defines common ways of working. According to Fowler (2006), some of them are maintaining a single source code repository, a high degree of automated tests, automated build process and as a result of following, always having a working, tested stable piece of software which could be deployed. (Fowler 2006.)

In addition, if approaching testing from a point of view that tests are written before the code (for example Test Driven Development, TDD), the tests help the team also with exploring the system design, not only with error detection. When it comes to system requirements and their verification, Continuous Integration approach targets to ensure that everyone involved with the project should have access to the latest state of the system, in

a form of a built executable that can be run. This makes it easier to adjust the requirements: if the people first see something that is not quite right, they can more easily state how the system needs to be changed. (Fowler 2006.)

2.3.2 Continuous Delivery

Continuous Delivery (often referred with abbreviation CD) extends Continuous Integration like practices to cover also the delivery of the software. For example, Chen (2015) describes Continuous Delivery as "software engineering approach, in which team keeps producing valuable software in short cycles and ensure that the software can be reliably released at any time". Chen explains that according to Continuous Delivery advocates, taking this approach would let organizations to bring service or product improvements to in reliable, rapid and effective manner, which leads to competitive advantage in rapidly changing environments. Based on his experiences of applying CD in practice he agrees with the possible advantage, but points out that adopting Continuous Delivery can be very challenging too. (Chen 2015: 50–51.)

When examining Continuous Delivery from requirements point of view, Chen presents that release models with only some releases each year artificially delay features that have been completed early in the release cycle. Based on his experience of delivering only few releases a year, receiving early feedback of the features was not possible and there was also a loss of value those features could have generated. In Continuous Delivery also user acceptance testing, which ensures that the system meets users' requirements, is done continuously throughout the development cycle and early feedback of new features should be available. (Chen 2015: 50–51.)

Despite the challenges to get started with Continuous Delivery, Chen presents that adopting the Continuous Delivery practices in his current organization has created them six main benefits. He characterizes these benefits as accelerated time to market, building the right product, improved productivity and efficiency, reliable releases, improved product quality and improved customer satisfaction. (Chen 2015: 52–53). Each of the observed benefits are presented in more detail in table 5.

Table 5 Six main benefits of adopting Continuous Delivery in large publishing company observed by Chen (2015: 52–53).

Accelerated time to market	When release frequency increased, it resulted also that cycle time from a requirement to market decreased from months to days.
Building the right product	Frequent releases have helped the development team to obtain user feedback faster, which helps them to be more confident that they are building the right product.
Improved productivity and efficiency	Chen thinks that efforts previously required to release activities and error fixing can be now used to more valuable purposes.
Reliable releases	Operational risk related to a release had reduced significantly, as majority of bigger problems have been discovered already during development.
Improved product quality	Chen states that the number of open bugs for the application have decreased almost 90%, and customers report almost none of them.
Improved customer satisfaction	Former tensions between teams caused by quality and release issues have relieved after introducing Continuous Delivery.

As the previous six benefits presented by Chen (2015) are based on his own observations and experiences related to adopting and implementing Continuous Delivery, those might not be applicable in all organizations. However, also DevOps employs Continuous Delivery in its production and delivery process.

2.3.3 DevOps

Ebert, Gallardo, Hernantes & Serrano (2016) describe DevOps (compound of abbreviations from Development and Operations) as an organizational cultural shift that integrates development, delivery and operations, targeting to flexible, fast development, fluid connection between teams and provisioning business processes. They present that the idea of

DevOps is to, instead of heavy and artificial process concepts, to make organizations focus on continuous delivery of small upgrades. As DevOps targets to high quality deliveries with short cycle times, Ebert et al. (2016) emphasize that high degree of automation and tools are mandatory in achieving this goal. They categorize commonly used DevOps tools covering areas such as build, continuous integration, deployment, logging and monitoring. (Ebert et al. 2016: 94, 96.)

Basing on examples like Google and Amazon, Ebert et al. (2016) present that achieved system cycle time of the can be even minutes, depending of the deployment model and constraints defined by the environment. They point out that tailored to the environment and product architecture, DevOps ideology is applicable to various different delivery models. However, they admit that same technology and models that work for example to web services do not apply to embedded software or safety critical systems. (Ebert et al. 2016: 94.)

Generic DevOps production and delivery process consists of requirements engineering, development, build management and deployment management. Requirements engineering owns the responsibility for feature mapping and dependency management. Development is done in feature teams and it employs Continuous Delivery practices. Both build and deployment management are also highly automated. (Ebert et al. 2016). This generic DevOps production and delivery process is presented in figure 2, visualizing also the relations between different operations.

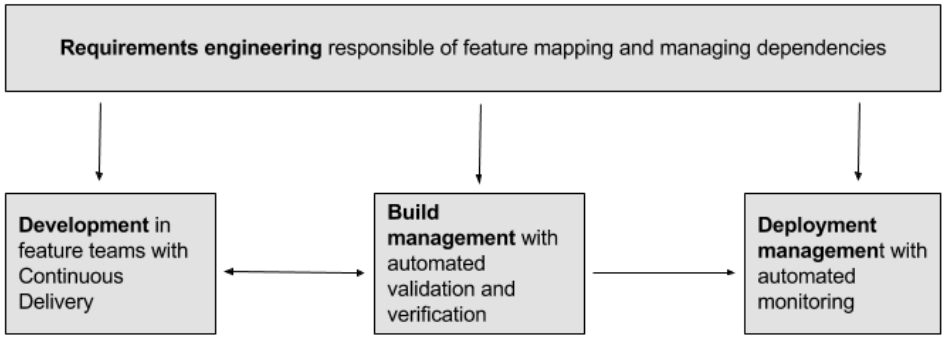

Figure 2 Generic DevOps production and delivery process presented by Ebert et al. (2016: 95).

As a conclusion, Ebert et al. (2016) argue that typically mutual understanding, starting from system requirements reaching to maintenance or product evolution, achieved with DevOps will improve the cycle time and reduce costs. These benefits result from fewer requirements changes, focused quality assurance and faster delivery cycles. When examining DevOps from system requirements point of view, also DevOps seems to offer the team early feedback of developed features and a possibility to adjust the requirements on fly. (Ebert et al. 2016: 99–100.)

3 REQUIREMENTS ENGINEERING IN SOFTWARE DEVELOPMENT

This chapter focuses on the theoretical background of requirements engineering in software development. Here we discuss about the purpose of engineering software requirements, the requirements process and activities, and some common ways of doing requirements engineering. By this, we seek to answer questions such as why software requirements are made and when requirements are defined. Also keeping our eye on the risk management perspective of this study, some common pitfalls on requirements engineering and building software are discussed. In this study, software requirement is defined according Hickey and Davis (2004: 72) as *any externally observable characteristic of a desired system*, which means the higher-level blueprint for the software building process.

3.1 The purpose of software requirements

Software engineering was initially defined by Boehm (1976: 1226) as the practical application of scientific knowledge to design and build software, and the required documentation to develop, operate and maintain the software. This documentation, defining the system requirements, specifying what the software should do and how it should behave, is one of the most essential parts of software engineering (Brooks 1986). However, defining the right system requirements is also one of the most error-prone and demanding tasks, or as Brooks (1986) has said: "The hard thing about building software is deciding what one wants to say, not saying it". Brooks (1986) explored the essence and common mistakes related to software development in *"No Silver Bullet - The Essence and Accidents of Software Engineering"* (1986) and even several decades his arguments have a valid point: the essence and accidents of software engineering are often related to the same factors, which are complexity and building the right thing. (Brooks 1986: 10–12). Also, Hickey & Davis (2004: 66) refer to Brooks (1986) as they emphasize the importance of requirements elicitation, and argue that poor requirements elicitation will most likely cause the final project to fail.

As the system level requirements define a basis and purpose for all later software development efforts, the importance of requirements engineering should not be underestimated. Brooks (1986) emphasizes that the most important function a software builder can perform to the client is iterative elicitation and refinement of system requirements. The software builder should also admit the fact that usually the client does not know what he wants, and extensive iteration of the design and requirements should be allowed. (Brooks 1986: 16–18). Admitting the same fact, client being unsure what kind of software system is actually needed, was also discussed in previous chapter with agile software development methods. For example, Fowler (2006) reasoned Continuous Integration practice by stating that if people first see a version that is not exactly what they want, it is easier to define how the system needs to be changed.

If following Brooks' arguments, the requirements engineering process and requirements specifications cannot rely only to the hopes placed upon test automation and other technological salvations on saving work and improving software quality. Brooks states, that for example even the most perfect test automation can only verify that the program is implemented according to its specification. Thus, the most essential and meaningful part in building software is to create a consistent and complete specification. Once such specification exists, much of the later programming work is just debugging the specification. (Brooks 1986: 11, 13, 16–18.)

3.2 Requirements process in software engineering

In this chapter, a generic description of a software engineering requirements process is discussed, as discovering, analyzing and defining the software system requirements is much more than just writing the requirements specification. As we already discussed why requirements engineering is done, in the context of our study it is still essential to understand when and how the software requirements are defined. Hickey & Davis (2004) point out, that no matter whether the software development approach is traditional with well-defined series of phases or iterative with repeated cycles of development activities, the

requirements will change throughout system development. According the traditional approach, the requirements activities should be performed at the beginning of the development process. In reality to keep on track with the ever-changing user needs, the requirements activities must be performed regularly. When using an iterative or agile development approach, the requirements activities should be performed at the beginning of each iteration. Despite the time point when performed, the requirements activities are a very essential for gaining the needed understanding of user needs and for the overall success of the software development effort. Thus, Hickey & Davis (2004) point out that understanding how the requirements process activities are performed is an important first step for improving any part of the process in industry. (Hickey & Davis 2004: 66–68.)

We view the requirements process according to definition of Hickey & Davis (2004) as series of five activities: elicitation, analysis, triage, specification and verification. When doing requirements work in reality, previous five requirement activities are performed in parallel and their proportions change over time in the requirements process. (Hickey & Davis 2004: 67). The five requirements activities are described in table 6.

Table 6 Requirements activities presented by Hickey & Davis (2004: 67).

Elicitation	Learning and discovering the customers', stakeholders' and users' needs
Analysis	Forming a list of candidate requirements and creating models of requirements by analyzing the knowledge elicited from stakeholders
Triage	Addressing subsets of requirements to specific releases of systems
Specification	Documenting the wanted features and external behavior of the system
Verification	Verifying the requirements consistency, completeness and reasonableness

The initial and often most critical activity in requirements process is elicitation, which targets to discovering the actual needs of the customers, stakeholders and user groups for whom the software system will be built. There is not only one way to do requirements

elicitation. Usually the analyst doing elicitation uses some technique, consciously or subconsciously, which could be at its simplest a discussion with stakeholders. Hickey & Davis (2004) present that the technique selection is driven by multiple factors: problem, solution, project environment characteristics and the state of the requirements. Common pitfalls related to elicitation technique selection are usually related to lack of experience of the analyst performing requirements elicitation. (Hickey & Davis 2004: 67–68.)

Based on previous observations, Hickey & Davis (2004) present four reasons why, alone or as a combination of previous reasons, the analyst usually selects the used elicitation technique. First, the selected technique might be the only one the analyst knows. Second, the selected technique is the analyst's favorite, no matter what is the situation. Third, the analyst follows a particular methodology, which suggests the used techniques. Fourth, the analyst understands that the selected technique is likely to be effective in the specific situation. Hickey & Davis also suggest, that if the elicitation technique is selected based on the fourth reason, it is most likely to provide best results of the elicitation. (Hickey & Davis 2004: 68–69.)

3.3 Approaches and techniques for requirements engineering

According to definition by Hickey and Davis (2004: 74), the term technique refers to a description of what to do, and it can also include suggested ways of doing it and suggested tools and notations to use. From the point of view of our study, suggesting the use of different requirements techniques is fundamental part of Continuous Requirements Risk Profiling method, when requirements risk is mitigated and resolved. The method applies the categorization presented by Mathiassen et al. (2007: 576), which divides the techniques into four groups: discovery techniques, experimentation techniques, prioritization techniques and specification techniques. In the tested method, previous categories are also mapped to certain risk categories for resolving the requirements risk.

3.3.1 Discovering requirements

Discovering the possible requirements is a natural and necessary starting point in requirements engineering. Discovering the requirements does not mean only getting a list of the wanted system features, but also understanding why those features would be needed. The discovery techniques emphasize the initial uncovering of requirements and focus on interactions with customers and would-be users. Many of these techniques target on creating a rich understanding of the user needs, finding out the motivation behind described needs and uncovering also such requirements that users find hard to describe. (Mathiassen & Tuunanen 2011: 41). Some techniques that can be used for requirements discovery listed by Mathiassen et al. (2007: 576) are for example: Brainstorming, Focus groups, Requirements workshops and Use cases.

3.3.2 Experimenting with requirements

When experimenting with requirements, user reactions and knowledge can be used to shape requirements by employing designs of the software artifact. The experimentation techniques focus on creating some model of the software artifact, presenting it to the audience and then learning and improving based on the feedback. These techniques can help to stabilize the requirements and to understand better the context where the system is going to be used. (Mathiassen & Tuunanen 2011: 41). Some techniques that can be used for requirements experimentation listed by Mathiassen et al. (2007: 577) are for example: Participatory design, Requirements prototyping, Testing and User-interface prototyping.

3.3.3 Prioritizing requirements

Requirements prioritization refers to the part of requirements engineering process, where the different project stakeholders analyze and negotiate which of the identified requirements should be chosen for implementation. Usually some metric is applied to the prioritization process to compare and decide which of the requirements are most valuable for

the designed system. Tuunanen and Kuo (2015) have recognized at least five different metrics for prioritizing the requirements: resources, performance, adaption, design and usability. The resource-based view considers factors such as cost, time, technologies and skills when prioritizing the requirements. (Tuunanen & Kuo 2015: 296–297.)

Even though some metric drives prioritization process, it is good to keep in mind that different people can prioritize the same requirements differently. Tuunanen and Kuo (2015) point out that different preferences, related for example to cultural factors, are not taken into account by most of the available requirements prioritization techniques. These cultural differences could be worth considering especially in cases when the software is developed to global markets with wide range of end-users. (Tuunanen & Kuo 2015: 296, 307.) Some techniques that can be used for requirements prioritization listed by Mathiassen et al. (2007: 576) are for example: Card sorting. Contextual design, Critical success factors and Quality function deployment.

3.3.4 Specifying requirements

Specifying the requirements aims to document the customers' and users' needs into either textual or graphical format. The specification techniques are documentation-centric, and used for creating the result we know as "requirements specification". Specification techniques also aim to bring suitable level of abstraction for the requirements, so that those can be communicated with and understood by different stakeholders and developers. (Mathiassen & Tuunanen 2011: 41). Some techniques that can be used for requirements specification listed by Mathiassen et al. (2007: 577) are for example: Data flow diagrams, Entity-relationship modeling and State charts.

3.4 Common challenges of requirements engineering and management

According to Brooks (1986) similar kind of concepts have and most probably will lead also to the accidents of software engineering. Important part of building software is fashioning complex conceptual constructions, and on the other hand also various different

kind of problems emerge from this complexity. In his opinion, four inherent software qualities causing many of the software engineering challenges are complexity, conformity, changeability and invisibility. (Brooks 1986: 11–12). Previous qualities are explained more detail in Table 7.

Table 7 The four qualities usually causing software engineering problems (Brooks 1986: 11–12).

Complexity	Various software engineering problems come from the inherent complexity and the way this complexity increases nonlinearly with software size. Besides technical problems, also management problems may emerge.
Conformity	Much of the complexity comes when the software must conform to other interfaces. Conformation to other interfaces creates such complexity that cannot be simplified by any redesign of the software system itself.
Changeability	Software systems are always embedded in diverse and complex cultural environments consisting of applications, laws, users and machines. As this environment changes continuously, it also inevitably forces changes upon the software system.
Invisibility	Reality of software is invisible, hard or even impossible to visualize, and not inherently embedded in space. The visualizations we try to create upon software are not univocal, and the same software structure can be visualized in numerous different ways. This already complicates the design process within one mind, but turns into even bigger problem when the structures need to be communicated with several minds across various stakeholders and development teams.

Correspondence to software complexity, software conformity various interface, environment changeability and software invisibility can be also seen in the in the Continuous Requirement Risk Profiling method. Thus, it could be hypnotized that also some of the requirements related risks and problems can originate from unsuccessful handling of previous problematic software qualities.

4 MANAGING RISK IN SOFTWARE PROJECTS

Over the decades that software systems have been built in hopes of business advantage and profits, also it has become clear that not all of the software projects are success stories. In this chapter, we discuss about the factors that can make the software development effort a failure and how to manage these risks. To gain a comprehensive understanding of the phenomenon, we seek to explain what software project risk and risk management actually is, why it is important and how the software projects could get started in identifying and managing different kind of risks.

Tiwana & Keil (2004) argue that most failing software projects have one common cause behind them, which is that the delivered system does not fit the actual problem. They also argue that good software is easy to recognize once one comes across it: good software solves the problem it was intended to and does what the users expect. Underperforming and failing software projects do not only cause headache and missed deadlines, but Tiwana & Keil (2004) also present measures that these projects result big financial loss to participating companies each year. (Tiwana & Keil 2004: 73–74). One recent example from Finland gaining a lot media attention has been medical company Oriola, facing losses worth millions of euros after switching into new information system (Kauppalehti 2017). Wallace et al. (2004) too emphasize that as the investments on software systems continue all the time, managing risk related to those projects should be also concerned.

Various different models, methodologies and tools have been developed to help the industry professionals to identify and manage these different types of risk related to software projects. A generic categorization of such tools is presented in the fourth subchapter. One such tools is also the Continuous Requirements Risk Profiling method introduced by Tuunanen et al. (2015), targeting to help industry professionals to identify the requirement related risk in their software projects. Continuous Requirements Risk Profiling method is introduced in the last subchapter.

4.1 What is software project risk and risk management?

Saarinen & Vepsäläinen (1993) present that software project risk can be summarized into two basic concepts: complexity and uncertainty. These are often related to factors such as size, stability and structure of the built software, difficulties to define the requirements, and participants' skills and previous knowledge of the technology. (Saarinen & Vepsäläinen 1993: 283–284). As the circumstances under which software systems are built have evolved over the decades, also new factors leading to different risk have been presented in research. One example is developing software for wide audience end-user, when the actual end-users might be hard to reach or unknown (Tuunanen 2003: 45–46). Another example is developing software in geographically distributing and large project teams, which can introduce problems such as language barriers, limited face-to-face interaction and time-zone differences (Persson, Mathiassen, Boeg, Madsen & Steinson 2009: 508).

In the context of software projects, Wallace, Keil & Rai (2004) define *risk* as a *"set of factors or conditions that can pose a serious threat to the successful completion of a software project"*. Often once identified, the goal is that these factors or conditions are managed, mitigated and their possible effects minimized. Consequently, risk management is described as managerial purpose to affect previous conditions. However, if understanding and managing software project risk fails, several kinds of problems can arise in the project. These problems can be for example unfulfilled user requirements, exceeded budgets and timelines and building systems that will not be used or that will not deliver value to its users. (Wallace et al. 2004: 115–116.)

Tiwana & Keil (2004) point out that even though not all events can be controlled, many of the common risks software projects face could be assessed and managed. They emphasize that for success, it is important to think software as a medium of knowledge rather than a product. When viewing software project risk from that perspective, the first thing to focus on would be successfully translating customer needs into system requirements and specifications. (Tiwana & Keil 2004: 73–74). Based on previous research, also Wallace et al. (2004) show that identifying and analyzing the threats is necessary so that some

actions reducing the risk can be taken. They argue that offering the project managers better information about the software project risk could help them to formulate risk management strategies aiming to mitigate the sources of most high risk. (Wallace et al. 2004: 116–117). Tuunanen et al. (2015: 4020) present that to manage the risk related to a software project the task that needs to be performed is assessing the project risk exposure.

4.2 Six dimensions of software project risk

As a target to offer the project managers better information about the possible dimensions and likely patterns of software project to help them formulate risk management strategies to mitigate the risks, Wallace et al. (2004) have created a categorization of the dimensions of software project risk. The six dimensions of software project risk defined by Wallace et al. (2004) are team risk, organizational environment risk, requirements risk, planning and control risk, user risk and complexity risk. (Wallace et al. 2004: 117). These dimensions with related characteristics are presented more detailed in table 8.

Table 8 Summarizing six dimensions of software project risk presented by Wallace et al. (2004: 117).

Team risk	Factors associated with the project team and its members, which can increase the uncertainty about the outcome of the project: team member changes, lack of sufficient knowledge among team members, cooperation or communication problems.
Organizational environment risk	Uncertainty surrounding the organizational environment where the project takes place: unfavorable organizational politics, instability or missing organizational support.
Requirements risk	Uncertainty related to the system requirements: frequent changes in requirements or incorrect, unusable or ambiguous requirements.
Planning and control risk	Uncertainty caused by poor project planning and control: unrealistic schedules and budgets, excessive schedule pressure or lack of visible milestones to estimate progress.
User risk	Uncertainty coming from lack of user involvement or communication during system development: users' unfavorable attitudes towards new system of lack of cooperation.
Complexity risk	Risk caused by the inherent complexity and difficulty to undertake the software project: use of unfamiliar technologies, high complexity of the processes being automated or many dependencies to existing or external systems.

Wallace et al. (2004) present that project characteristics such as project scope, the degree on the project is strategic and possible outsourcing of software development efforts, also have their own impact to the risk level. These three characteristics, besides several other characteristics which were left out of the scope of their study, are presented to lead to different dimensions of risk. Wallace et al. (2004) present that the characteristics related to strategic orientation of the project are likely to lead to project complexity risks, char-

acteristics related to scope of the project can lead to all risk dimensions, and characteristics related to degree of outsourcing can lead to team risk and planning and control risk. (Wallace et al. 2004: 120–122). The relationships between presented project characteristics and different types of risk revealed by Wallace et al. (2004) are presented in figure 3.

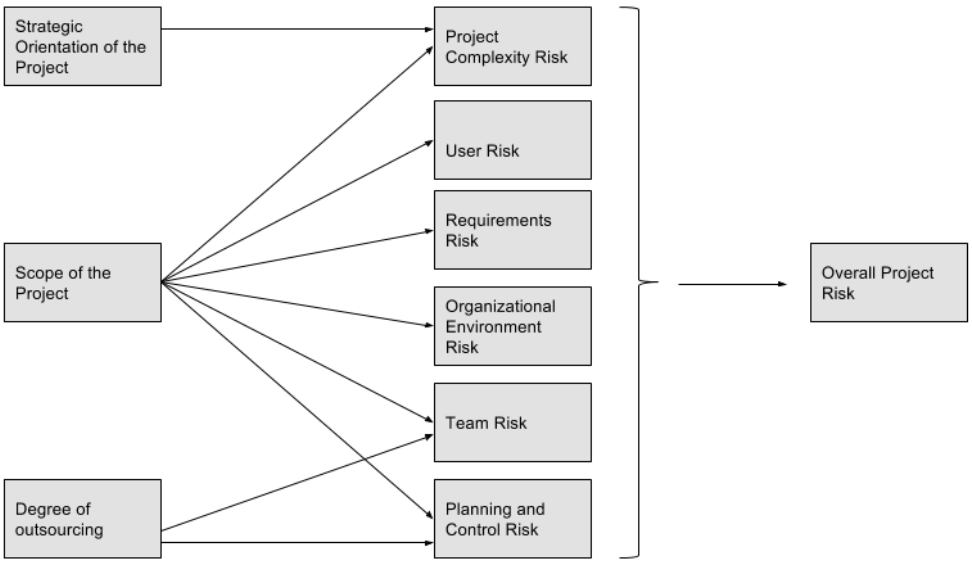

Figure 3 The relationships between presented project characteristics and different types of risk revealed by Wallace et al. (2004: 121).

Basing on the empirical evidence of their study, Wallace et al. (2004) also argue that the most notable risks in high-risk projects are often different to the ones observed from medium risk and low risk projects. Their results show that for high-risk projects, the most notable are requirements risk, planning and control risk, and organizational environment risk. For low risk projects, complexity risk is the most notable one. From requirements risk management point of view, the previous results suggest that in high-risk projects managing requirements related risk is important for controlling the overall risk level. (Wallace et al. 2004: 120, 122.)

4.3 Focus on software project requirements risk

In this subchapter software project requirements risk, one of the six dimensions of software project risk by Wallace et al. (2004), is discussed in more detail. The introduced view of requirement risk is based on the publication "*A Contingency Model for Requirements Development*" by Mathiassen et al. (2007), which also serves as the theoretical framework for the method tested in our study.

When it comes to software project requirements development, Mathiassen & Tuunanen (2011) present that a typical starting point for the requirements process can be just an informal presentation of vague ideas. For later success of the development project, it is then highly important these vague ideas evolve into formalized requirements, which serve as a guideline for the system design and implementation. Mathiassen & Tuunanen (2011) also point out that in past decade IT projects have took a step into direction where the system's users might be either unknown or out of direct reach, which brings its own challenges to requirements management. This can be the situation when developing for example a new kind of mass-market software. In both cases, acknowledging and managing the requirement related risks is an important success factor. (Mathiassen & Tuunanen 2011: 40.)

When considering how to mitigate the requirement related risks, Mathiassen & Tuunanen (2011) suggest to first approach requirements risk management with three preparative steps:

1. Identifying the risk types
2. Organizing a toolbox
3. Integrating risk management practices

The first step and initial step is identifying the risk types (Mathiassen & Tuunanen 2011). As stated already before, identifying and recognizing possible risk factors is the initial step for managing any type of risk (Wallace et al. 2004). The second step is organizing a

toolbox the risks can be mitigated with. The third and final step is integrating risk management practices to work together with requirement risk management. (Mathiassen & Tuunanen 2011: 40–42.)

4.3.1 Identifying and categorizing the requirements risk

The starting point for risk analysis and management is to identify the requirements risk types that characterize the concerned project(s). Initially Mathiassen, Tuunanen, Saarinen & Rossi (2007) suggested dividing requirements risk into three types: identity, volatility and complexity. Later Tuunanen et al. (2015: 4027) presented fourth complementary category, requirements integrity risk. These categories contain also some similar characteristics, and Mathiassen et al. (2007) present that both identity and complexity risks imply elements of poorly understood requirements. Summary of four requirements risk categories is presented in table 9. (Mathiassen et al. 2007: 574.)

Table 9 Summary of requirements risk categories introduced by Mathiassen et al. (2007) and later complemented by Tuunanen et al. (2015: 4027).

Requirements identity risk	Requirements availability: if exposed, indicates that requirements are unknown or indistinguishable.
Requirements integrity risk (Tuunanen et al. 2015)	Requirements completeness and accuracy: if exposed, indicates that there are difficulties to understand the origin and relations between identified requirements.
Requirements volatility risk	Requirements stability: if exposed, indicates that requirements change easily as a result of environmental changes or individual learning.
Requirements complexity risk	Requirements understandability: if exposed, indicates that requirements are hard to understand, specify or communicate.

Requirements identity risk is likely to be discovered in situations when there is communication gap between the software system developers and the future users. The risk is

related to the poor availability of the requirements, and thus a high requirements identity risk indicates that the actual system requirements are indistinguishable or not known by the developers. Mathiassen et al. (2007) present that this risk reflects the distance between the developers and the users: this distance can be physical, cultural or conceptual. The distance is likely to appear for example in situations when mass-market software is developed, and the actual end-users are unreachable or unknown. (Mathiassen et al. 2007: 574–575.)

Requirements volatility risk refers to the instability of the requirements: such uncertainty appears when the requirements change easily because of environmental dynamics. Mathiassen et al. (2007) presented that these dynamics are related to the stakeholders' individual learning during the software development process, driven either by changes in internal or external conditions. Mathiassen et al. (2007) highlight that the literature presents that requirements process is often dialectic and defining the requirements often reveals new options for the stakeholders. In addition, the users' needs usually are not self-evident for the developers. (Mathiassen et al. 2007: 574–575.)

Requirements complexity risk is related to the understandability and the ease of communication of the requirements. Uncertainty related to requirements complexity appears when the requirements are difficult to communicate, specify and understand. This is one of the inherent features of software, stated also by Brooks (1986). Tuunanen et al. (2007) present from literature that additional sources for complexity may appear from the requirements process, when many and possibly conflicting stakeholders' views about one software system are put together. (Mathiassen et al. 2007: 574–575.)

Requirements integrity risk was later added by Tuunanen et al. (2015) based on the empirical verification of the theoretical model. Requirements integrity refers to the completeness and accuracy of the elicited requirements. High requirements integrity risk refers to problems in understanding where the requirements have originated. (Tuunanen et al. 2015: 4027.)

4.3.2 Resolving requirements risk

The second step towards effective requirements risk management is to organize an appropriate set of tools that can be used for mitigating the risks. To summarize, once the requirements risks have been identified, resolving the risks means improving the current requirements from those aspects that could pose a risk for the software project. Techniques and tools are the same that can be used when the requirements are initially defined, now those are just applied to improve the requirements in such way that mitigates possible risks. Mathiassen & Tuunanen (2011) present that most probably some suitable techniques are already in use, but they suggest evaluating each technique's effectiveness to the risk types and complementing the toolbox on based on what is missing. (Mathiassen & Tuunanen 2011: 41). The four categories of requirements techniques by Mathiassen et al. (2007) were already discussed more detailed in chapter 3.3, and are summarized here in Table 10.

Table 10 Requirements development technique types (Mathiassen et al. 2007: 576).

Requirements discovery	User- and customer-centric. Facilitate identification, learning and prediction of the user needs.
Requirements prioritization	Resource-centric. Target on analysis, assessment and selection between already identified requirements.
Requirements experimentation	Software- and solution-centric. Use user reactions and knowledge to shape requirements by employing designs of the software artifact.
Requirements specification	Documentation-centric. Use abstraction and graphical or textual representations to provide explicit and agreed upon requirements for further development.

4.4 Risk management methods and tools

Several researchers emphasize the importance of software development professionals' risk management related knowledge behind successful risk management. Tiwana & Keil (2004) argue that only such risks that have not been taken into account and are unmanaged will have the power to surprise during the project execution. They believe that if the project managers understand the factors that drive risk and which of them can be influenced, the project managers could better accept the ones that cannot be changed and also have the courage to manage the risks that are in their control. (Tiwana & Keil 2004: 77.) For these purposes of identifying and influencing the factors related to risk, various different kinds of risk management tools have been developed. Table 11 presents a generic classification for available risk management tool types synthetized by Iversen et al. (2004) and later analyzed by Mathiassen, Saarinen, Tuunanen & Rossi (2004: 35–36).

Table 11 Summary of four types of risk management models and their characteristics identified and synthesized by Iversen et al. (2004) and later analyzed by Mathiassen et al. (2004: 35–36).

Risk lists	Present a prioritized list of potential risk items, but no resolution actions are suggested.
Risk-action lists	Present a prioritized list of potential risk items and suggest one or more resolution actions to mitigate each exposed risk item.
Risk-strategy models	A contingency model, mapping synthesized common risk profiles to synthesized common resolution action patterns.
Risk-strategy analysis models	A stepwise process identifying potential risk items, linking them to resolution actions, and forming an overall risk management strategy.

To gain a better overall understanding of the available risk management tools, we use "The One Minute Risk Assessment Tool" by Tiwana & Keil (2004) as an example of tool for managing the overall software project risk. In their tool, Tiwana & Keil (2004) classify two kinds of knowledge, which they see as essential for building any kind of software

system: technical knowledge and customer knowledge. To develop the tool, they examined six key risk drivers from previous two categories which determine how well that essential knowledge can be embodied. The categories with related risk drivers are presented in table 12.

Table 12 Six software project key risk drivers presented by Tiwana & Keil (2004: 75).

Embedded knowledge	Lack of customer involvement Dissimilarity to previous projects (technical) Requirements volatility
Execution coordination	Use of an inappropriate methodology Lack of formal project management practices Project complexity (coordination)

Based on the result of their study which examined 720 different software projects, Tiwana & Keil (2004) assigned weights to each risk driver based on their likely relative impact on the project outcome. Thus, at its simplest form, by examining six key risk drivers the industry professionals could make a rough estimate on their project risk level from low to high risk. (Tiwana & Keil 2004: 75–77.) The tool by Tiwana & Keil (2004) could be categorized to represent a risk list among all risk management tools, which only represents an assumption of the potential risk level and does not provide any solution proposals.

4.5 Introducing the Continuous Requirements Risk Profiling Method

The Continuous Requirements Risk Profiling Method tested in following parts of this study has been developed by Tuunanen, Vartiainen, Ebrahim & Liang (2015) and the development process has been presented this far in conference publication "*Continuous Requirements Risk Profiling in Information Systems Development*". The presentation of the method as it is tested and examined with the industry professionals in this study is

still unpublished and referenced here as Tuunanen et al. (2016). The method was initially developed by using design science research as a research method, and the study was conducted in cooperation with the Project Management Institute of New Zealand. In this phase industry experts were involved in the development of initial method by focus groups interviews and a Delphi survey. (Tuunanen et al. 2015: 4019.)

Tuunanen et al. (2015) got motivation for developing the method of gap they had noticed between how the literature views risk management and how the increasingly popular continuous and agile software development approaches that move from release to release. In the developed method targets to take into account the iterative and repeated nature of requirements, design and implementation phases that appears for example in agile and lean software development approaches. Tuunanen et al. (2015) present from literature that for the risk management to be effective with iterative software development approaches, it needs to be iterative too as presented in Figure 4. As requirements have a crucial role in the potential success software development efforts (Boehm 1976: 1227), Tuunanen et al. (2015) argue that requirements risk should be also the driver for the whole software project risk. (Tuunanen et al. 2015: 4019–4022.)

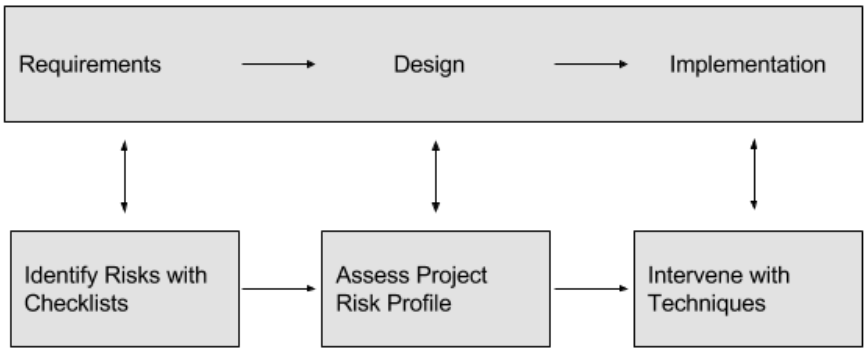

Figure 4 Applying continuous requirements risk profiling and management in continuous development (Tuunanen et al. 2015: 4020).

The developed method focuses on requirements risk and is based on the previously introduced theoretical model by Mathiassen et al. (2007). In their study, Tuunanen et al. (2015)

took the first step in developing the Mathiassen et al. (2007)'s conceptual and literature-based framework into a method that could provide assistance for industry professionals. First Tuunanen et al. (2015) validated the theoretical framework with inputs received from focus group interviews conducted with industry professionals. In this first part of their study, they found a positive match between the risk categories presented in literature and the types of risks professionals face in practice. According to Mathiassen's presentation, the method classifies the requirement risk in three categories: requirements identity, requirements volatility and requirements integrity. Tuunanen et al. (2015) found also one additional category that was added to the method, requirements integrity. The focus group interviews resulted the initial checklist items, later refined, reviewed and complemented in the expert panels in Delphi study rounds. The two Delphi study rounds were used for defining the phases each risk item would be most likely to affect to and the indicative risk impact levels presented in the risk-profiling table. The risk items were organized according to the panel's suggestions after a reasonable agreement had been achieved. (Tuunanen et al. 2015: 4021–4025.)

Table 13 The initial risk resolution pattern presented by Tuunanen et al. (2016).

1. **Identify risks** with checklists for each ISD phase. Nominal process starts with requirements phase and continue to design and implementation phases.
2. **Assess project risk profile** by recognizing individual requirements risks affecting the project. Use the indicative impact levels to prioritize requirements risks.
3. **Intervene with Requirements Risk Resolution Techniques** according to the risk resolution rules in following order using the appropriate techniques:

- If identity risks are high, put high emphasis on discovery techniques.
- If integrity risks are high, put high emphasis on prioritization techniques.
- If volatility risks are high, put high emphasis on experimentation techniques.
- If complexity risks are high, put high emphasis on specification techniques.
- If three or more risk items are high, follow the above sequence of applying techniques (from 1 to 4).

For the first step of Continuous Requirements Risk Profiling method, "Identify risks", Tuunanen et al. (2015) provided three checklists, presented in tables below: *Requirements phase checklist* (Table 14), *Design phase checklist* (Table 15) and *Implementation phase checklist* (Table 16). For the second step, "Assess project risk profile", Tuunanen et al. (2015) provide a risk-profiling table (Table 17), presenting the suggested indicative impact for each risk item in specific development cycle phase.

Table 14 The initial requirements phase checklist by Tuunanen et al. (2015).

Risk	**Risk Type**	**Project is/can be exposed?**
Absence of Project Sponsor	*Identity*	
Access to Clients (Proximity to Source)	*Complexity*	
Ambiguous Requirements	*Identity*	
Change in Business Strategy and Direction	*Volatility*	
Change in External Regulations	*Volatility*	
Client Commitment	*Identity*	
Constrained Users' Knowledge	*Complexity*	
Fixed Budget and Timelines	*Integrity*	
Incorrect Stakeholder	*Identity*	
Misunderstood Business Needs	*Identity*	
Underestimation of Change Magnitude	*Volatility*	
Unrated Requirements	*Volatility*	
Any other risks that could affect design and implementation		

Table 15 The initial design phase checklist by Tuunanen et al. (2015).

Risk	**Risk Type**	**Project is/can be exposed?**
Ambiguous Requirements	*Identity*	

Change in External Regulations	*Volatility*	
Client Commitment	*Identity*	
Compliance with External Regulations	*Identity*	
Conflicting Requirements	*Integrity*	
Missing Requirements	*Identity*	
Delivering What the Client Requires	*Identity*	
Emerging Requirements Dependency	*Volatility*	
Fixed Budget and Timelines	*Integrity*	
Knowledge Gap between Coworkers	*Complexity*	
Lack of Collaboration	*Complexity*	
Technology Changes	*Volatility*	
Underestimation of Change Magnitude	*Volatility*	
Unrated Requirements	*Volatility*	
Any unresolved risks from requirements and risks that could affect implementation		

Table 16 The initial implementation phase checklist by Tuunanen et al. (2015).

Risk	**Risk Type**	**Project is/can be exposed?**
Ambiguous Requirements	*Identity*	
Change in External Regulations	*Volatility*	
Client Commitment	*Identity*	
Fixed Budget and Timelines	*Integrity*	
Hostile Users	*Identity*	
Project Team Member Turnover	*Volatility*	
Unrated Requirements	*Volatility*	
Underestimation of Change Magnitude	*Volatility*	
Any unresolved risk items from design and requirements		

Table 17 The initial risk-profiling table by Tuunanen et al. (2015).

Requirements Phase Specific Risks	**Impact**	**Design Phase Specific Risks**	**Impact**	**Implementation Phase Specific Risks**	**Impact**
Absence of project Sponsor	High	Missing requirements	High	Hostile users	Medium
Access to clients (proximity to source)	High	Delivering what the client requires	High	Project team member turnover	Medium
Incorrect Stakeholder	High	Compliance with external regulations	Medium		
Misunderstood business needs	High	Conflicting requirements	Medium		
Change in business strategy and direction	Medium	Emerging requirements dependency	Medium		
Constrained by users' knowledge	Low	Knowledge gap between coworkers	Medium		
		Lack of collaboration	Medium		
		Technology changes	Low		
Risks Affecting All Phases					
Ambiguous requirements	High	Ambiguous requirements	High	Ambiguous Requirements	High
Unrated requirements	High	Unrated requirements	High	Unrated Requirements	High
Client commitment	High	Client commitment	High	Client Commitment	High

Change in external regulations	Medium	Change in external regulations	Medium	Change in external regulations	Medium
Underestimation of change magnitude	Medium	Underestimation of change magnitude	Medium	Underestimation of change magnitude	Medium
Fixed Budget and Timelines	Medium	Fixed budget and time lines	Medium	Fixed budget and timelines	Medium

For the third step, “Intervene with Requirements Risk Resolution Techniques”, Tuunanen et al. (2016) adapted the risk resolution techniques listing and categorization from Mathiassen et al. (2007) to provide suggestions how the identified and prioritized requirements risks could be resolved. The listing contains full results of related literature review and is provided as appendix of this study. Some examples of the techniques were already presented in chapter 3. The Continuous Requirements Risk Profiling method in the previously des, is tested and improved in the later parts of this study.

5 RESEARCH PROCESS AND METHODS

In this chapter, the theoretical background of the used research process and methods is introduced. First using interpretive case studies in information systems research is discussed. As an outside observer researcher role was chosen for the case study, it suggests that the main data collection method for the study is likely to be interviews. Thus, we next introduce theme-centered interviews as a data collection method. As the data, we want to collect in our study is mainly qualitative, we discuss about qualitative data analysis based on themes. After presenting the possible research process and methods, we evaluate the validity and reliability of the study conducted with the presented approach. Finally, the design of the study for testing and improving the Continuous Requirements Risk Profiling method is presented.

5.1 Interpretive case studies

The interpretive case study approach presented in this research bases on the publication "*Interpretive case studies in IS research: Nature and method*" by Walsham (1995), which describes how case studies can be used for examining the software development related issues. Walsham (1995) states that the social issues related to software development has led to adoption of empirical methods to examine the related human meanings and interpretations. Common tool for this are interpretive case studies, where the researcher collects observations from the field over longer period. These interpretive case studies have their philosophical basis on ethnographic research tradition related to organizational research. (Walsham 1995: 74–75.)

Theory serves usually as the initial framework for the study, but the researcher should avoid seeing only what the theory suggests and explore also potential new issues. The role of the researcher in these studies can be either outside observer or involved researcher. Researcher with outside observer role maintains some distance to the people in field organization, and conducts the research by influencing the research domain only by sharing concepts and interpretations with the related people. Involved researcher will be

a member of the group or organization who actively participates the issues related to research domain. The choice of role should be consciously made in an explicit and well-reasoned way. The choice should be also reported with the results. (Walsham 1995: 76–77.)

For an outside observer researcher, an interview is usually the primary source of data as that is the best way to access the interviewees' interpretations on the research topic. Also for the involved researcher, interviews will provide a valuable way to examine the fellow participants' interpretations in detail. Directing the interviews too closely should be avoided to achieve a rich set of data for the interpretations, and the interviews recorded to extract useful set of data from those. Besides the right interview technique, access to people's thoughts, interpretations and views also requires good social skills from the researcher. (Walsham 1995: 77.)

Reporting the interpretive case study should be done in enough detail and some credibility achieved, as the study reports interpretations of people's experiences. Reporting should cover description of research sites, reason of choice, number of interviewed people and their professional position, possible other data sources and the period of time when research was conducted. When employing the interpretive case study approach, the limitations related to generalizability of the results should be concerned. The results of such study are explanations of particular phenomena, related to specific settings. Despite the criticism that results and interpretations cannot be generalized directly to be applicable to all contexts, those may valuable in future also in other contexts and organizations. (Walsham 1995: 78–79.)

5.2 Theme-centered interview as a research method

In our interpretive case study, the outside observer role was a likely choice for conducting the research. This suggests that the main data collection method is interview, described in more detail in this chapter.

Interview in its basic form, is interaction between the interviewer and interviewee. Interview has different objectives than an informal conversation: An interview aims to collect reliable information related to certain topic. In research interviews, the collected information is later used for solving some practical research problem, after analyzing and summarizing it with scientific methods. Typically, research interview is characterized by features such as planning some parts of the interactions beforehand, the researcher's previous knowledge on the topic, the interview situation is guided and by the interviewer and confidentiality between the interviewer and interviewee. (Hirsjärvi & Hurme 2000: 42–43.)

Hirsjärvi & Hurme (2000) present an interview method called theme-centered interview (teemahaastattelu) in their publication "*Tutkimushaastattelu: Teemahaastattelun teoria ja käytäntö*". Theme-centered interview is an approach to conducting semi-structured (often also referred as semi-standardized) interviews. In general, semi-structured interviews refer to research interviews, which have some standardized parts of viewpoints that are same in all conducted interviews, but otherwise the interviewer uses communication to tailor the interview depending of the situation. In semi-structured interview, the standardized part can be for example the types of the questions or the topics to be discussed about. (Hirsjärvi & Hurme 2000: 47–48.)

The concept of a theme-centered interview focuses on each interviewee's individual thoughts, experiences and beliefs. The method highlights how the interviewees experience the situation and what kind of interpretations they make based on it. Theme-centered interview itself does not force the results to be analyzed either with qualitative or quantitative approach. The most fundamental idea in theme-centered interview is that instead of presenting standardized set of detailed questions, the interview proceeds based on some central themes. Hirsjärvi and Hurme (2000) present that this separates the interview from the interviewer's control, and presents the voice of the interviewees. The approach considers the people's interpretations of situations. Besides the interpretations, the meanings associated with things and phenomena are in fundamental role. The open communication and interaction facilitated in a theme-centered interview is presented as an important factor for discovering previous factors. (Hirsjärvi & Hurme 2000: 48.)

Generic process for conducting theme-centered interviews according to Hirsjärvi & Hurme (2000) is presented in figure 5. Theme-centered interview process covers preparations such as designing the study, actual interviews and analysis of the collected interview data.

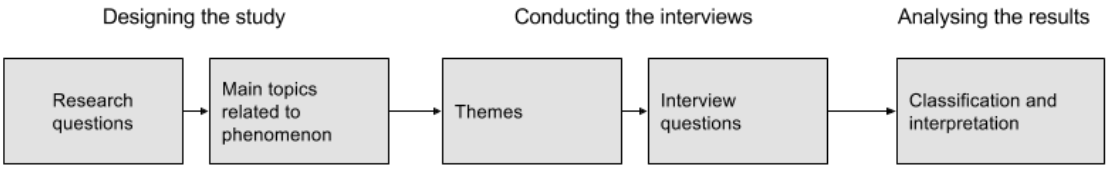

Figure 5 Theme interview process presented by Hirsjärvi & Hurme (2000: 67).

5.3 Qualitative data analysis based on themes

Thematic analysis is a qualitative method that helps to identify, analyze and report patterns, referred as themes, within data. When applied, it minimally describes and organizes the analyzed data set. Despite the variety of qualitative methods is vast, thematic analysis is presented as a foundational method for qualitative analysis. (Braun & Clarke 2006: 78–80). The same approach, less rigorously defined, is referenced for example as thematizing by Hirsjärvi & Hurme (2000: 173). According to Braun & Clarke (2006), qualitative methods can be positioned into two categories based on their dependence or independence of certain theory and epistemology. Thematic analysis represents the approaches that are independent of theory and epistemology, and is thus applicable to wide variety of problems. Thematic analysis is described as easily accessible also for researchers having less experience in qualitative research, as thematic analysis does not require such detailed theoretical and technological knowledge as other qualitative approaches. (Braun & Clarke 2006: 77–79.)

Braun & Clarke (2006) presented also a six-step guide for performing thematic analysis in a defined and organized way. The steps with their description are presented below in table 18.

Table 18 Six steps of thematic analysis presented by Braun & Clarke (2006: 87).

Step	Description
Familiarize with the data set	Transcribing, reading and re-reading the interviews.
Initial coding	Finding interesting features across the data set and labeling them with codes, collating related data together.
Searching for themes	Gather similar or related codes together into candidate themes. Theme should represent shared meaning.
Review candidate themes	Reviewing the themes in relation to the coded features and the full data set. Form a thematic map of the reviewed themes.
Define and name themes	Refining theme features and meanings, creating distinct naming and description of each theme.
Reporting the analysis	Forming a report of the analysis, presenting the results with example extracts of each theme and showing how these extracts and themes relate back to the research questions and literature.

Besides performing the steps for qualitative analysis steps, certain questions related to the analysis need to be considered and reported explicitly. These are how theme is defined, whether the analysis describes the whole data set or some certain aspect, is the analysis inductive or theoretical, are the themes semantic or latent, and if some epistemology is applied. (Braun & Clarke 2006: 81–85.)

For less rigorously defined thematizing approach, Hirsjärvi & Hurme (2000) present that if the data has been collected based on theme interviews, it can be also analyzed based on themes applying several different qualitative analysis techniques. Thematizing also starts by transcribing, classifying and coding the data set. Hirsjärvi & Hurme (2000) present three approaches for coding and classifying the transcribed data set, based on the tools that available for the research. Classification can be done either by using computer and a text analysis software, using computer without a specific software, or by performing the

analysis by hand using cards. In the case when analyzing the data with computer without a specific analysis software, Hirsjärvi & Hurme (2000) suggest addressing classes to the transcribed data set and formulating those into themes with functionalities available in regular text editors. From classified data, different kind of approach might be needed if the themes will be identified among all of the interviewees or separately from each interview. (Hirsjärvi & Hurme 2000: 141, 147, 172–173.)

Also in thematizing, the analysis consists of describing the data set, coding and classifying, finding associations and finally reporting the results of the analysis. Codes and classes applied to the data set create the framework, based on which the actual interpretations of the data set can be made. The classes should have both conceptual and empirical justification. Finding associations within the data set means discovering recurring patterns, rules and similarities either between the interviews or within one interview. Usually some deviation within the results is found as well. Thematizing can be complemented with other qualitative analysis tactics, such as counting or assigning data to scales. Counting refers to simply counting how many times certain factor, feature or phenomenon appears in the data set. Assigning data to scales refers to using for example nominal or ordinal scales to compare the observations of the data set based on certain feature. (Hirsjärvi & Hurme 2000: 143, 147–149, 172–176.)

5.4 Validity and reliability of the study

As discussed earlier, especially the limitations related to generalizability and repeatability of the results should be concerned if an interpretive case study approach is chosen (Walsham 1995: 79). The interpretive case study approach sets limitations to the applicability and repeatability of the results, and the results mainly apply to the context where the study was conducted. However, we find this approach with previously described research methods suitable for providing one aspect on feasibility of Continuous Requirements Risk Profiling method in agile software projects and collecting valuable feedback from professionals working in industry. This also allows us to provide an improved version of the

model suitable to the case company environment and settings, and thus provide an empirically validated and valuable tool for managing requirements risk in industry.

Another aspect related to the validity and reliability of the study concerns using interviews as primary data collection method. As the interviews are analyzed with qualitative methods and the analysis is mainly interpretive, the limitations should be noted when considering the repeatability of the study. When taking into account the case company environment the interviews are likely to be in both Finnish and English as the interviewees will represent several different nationalities, and some parts of the interview data needs to be translated by the author for the analysis process. Thus, possible errors on translating and interpreting the interviewees' original thoughts should be also considered.

5.5 Designing the study

This subchapter describes how the study was designed, which assessment criteria was used as a basis of the interview questions, how the interview questions were validated to be suitable for the planned study and what kind of structure was planned for the interviews.

The study was conducted during summer and autumn 2017. Some preparations and base study related to the topic and thesis seminars were done already on autumn 2016, but the actual literature review took place on spring 2017, interviews on summer 2017 and analysis on autumn 2017. The interviews were conducted when author was working in the case company.

5.5.1 Description of the case company

The case company of this study is a medium-sized software company based in Finland, which offers various types of software development services, mainly focusing on software project subcontracting. The case company's professionals have wide technological

expertise. Interviewees participating to this study represented a comprehensive range of the different types and sizes of projects, implemented to several domains. Agile project management practices are used widely across the case company, and the organizational structure could be described as lean and flat.

5.5.2 Purpose of the interviews

The objective for the research interviews conducted is to collect feedback and insights about the continuous requirements risk profiling method from industry professionals working in agile and DevOps software projects. The interview questions are designed to collect empirical data that could provide answers to the research questions, which means that how well the developed method works in practical company environment, do the industry professionals find the method useful and how the method could be improved to better correspond their needs. The interviews are carried out as theme-centered interviews and results analyzed with qualitative methods.

5.5.3 Applied assessment criteria

Assessment criteria for the method are applied from MIS Success Measures presented by DeLone & McLean (*Information Systems Success: Quest for the Dependent Variable*, 1992). In this case, the MIS Success Measures are applied to assess the Continuous Requirements Risk Profiling method. Of the original set of MIS Success Measures presented by DeLone & McLean (1992: 84–85), the ones that were best applicable for evaluating the method were chosen. The chosen MIS Success Measures were then used to derive and formulate the interview themes and suggested interview questions, so that the questions aim to seek answers about the method quality, method information quality, method information use, user satisfaction, individual level impact of the method use and possible project level impact of the method use. The selected MIS Success Measures and how they are applied as assessment criteria for the Continuous Requirements Risk Profiling method are presented in table 19.

Table 19 Assessment criteria adapted from MIS Success Measures presented by DeLone & McLean (1992: 84–85).

Category	**Appliance in this study**	**Chosen criteria**
System Quality	The method should be applicable and beneficial in practical project work	*Efficiency*: ease of use, ease of learning, method usefulness *Flexibility*: applicability to different kind of projects, possibility to modify the model
Information Quality	The method should give useful information about the requirements risks	*Understandability (content)*: the method is easy to understand and presented using language that is familiar for the professionals working in industry *Completeness and accuracy*: the information in method is complete and accurate enough
Information Use	The method should support the information use together with other systems	*Report acceptance*: the method is compatible to the reports/information already available *Motivation and voluntariness to use*: industry professionals feel motivated and voluntary to adopt the method as a part of their tool set
User Satisfaction	The method should meet its user's needs and expectations	*Information satisfaction (needed vs. received)*: information provided by method meets the users' needs and expectations *Decision-making satisfaction*: the information provided by the method are complete and accurate enough to support decision making
Individual Impact	The method should provide the needed support for project personnel managing requirements risk	*Accurate interpretation*: the method suggests accurate interpretations of the project situation *Decision effectiveness*: the interpretations suggested by the method are correct and would contribute to project decision making
Organizational Impact	The method should improve project decision-making and eventually IS success	*Overall project productivity gains*: the method use could have positive impact to project team productivity *Improved outputs and decision-making*: the method use could result as improved outputs of the requirements process, and eventually improve project success due to improved requirements risk control

From these criteria, we see that our data collection method can give input to mainly criteria 1–5. Criterion 6 is harder to measure, as organizational level impact can be estimated only on project, not on company level. Also on project level, our interviews can capture only one perspective of the possible project level impact. Questions related to criterion 6 are also presented in this plan as a part of interview theme 4, but should be considered in each interview if those are applicable and could give reliable results in particular situation.

5.5.4 Planning and validation of the data collection method

Before starting the study in the case company, both the study design and interview questions were validated with research group representatives and with a case company representative. These validations targeted to make sure that the interview questions were understandable, would collect enough empirical evidence for the purposes of this study, and to find out if the professionals working in the case company are likely to be familiar with our research topic.

First, the interview structure and questions were validated with the instructors Tuunanen and Vartiainen. The purpose of this validation step was to make sure, that the data collection method could provide information that is feasible in answering the research questions. After the approval of Tuunanen and Vartiainen, the interview structure and questions were validated with the case company representative. The purpose of this validation step was to make sure that the planned interviews were applicable to the organizational settings and environment of the case company. During this validation step, it was noted that the industry professionals working in case company were unlikely to be familiar with the theoretical framework of our study. Thus, a theoretical introduction was added to the beginning of the interviews to lead the conversation to the right topic and to gain the best benefit of the interviews for the both parties.

For the theme-centered interviews conducted for this study, a qualitative approach was chosen. The central themes to be discussed in the interviews were closely related to the

research questions. The themes chosen to interviews were getting familiar with the context and background of the case project to be discussed about, applying the tested method in case project, assessing the method completeness, accuracy and understandability, and assessing the method feasibility to project use and usefulness. Differing of some theme interview approaches, the interview questions were also formulated before the interviews. However, all the questions were not mandatory and were used more to facilitate the interaction. The interview was usually tailored to the situation, and some additional questions were presented to the interviewees based on their answers.

5.5.5 Structure of the interviews

The structure of the interviews was formulated around the concept of theme-centered interviews, taking into account structure of the tested Continuous Requirements Risk Profiling method and the applied assessment criteria. In addition, the needed introduction to the theoretical framework, discovered in second validation step of the data collection method, was added to the beginning of the interview structure. The planned structure for the theme-centered interviews is presented in table 20.

Table 20 . The planned structure for the theme-centered interviews.

Introduction of the study
Introduction to requirements risk management method background
Theme 1: Introducing the interviewee and the case project requirements
Theme 2: Assessing and prioritizing case project risk profile using the method
Theme 3: Discussion about method completeness, accuracy and understandability
Theme 4: Discussion about method usefulness and feasibility to project use
Summary and ending

During the introduction part, the interviewer and research topic was introduced to the interviewee. In addition, the common interview practices were described and the interviewee was made conscious about the purpose of the interview: testing feasibility of the developed Continuous Requirements Risk Profiling and prioritization method in a case project.

Before starting the actual interview, some theoretical background related to the research topic was presented to the interviewee. When testing the interview questions with a case company representative, it was noticed that presenting the theoretical background for the interviewee would help to lead the conversation to right direction. Introduced theory is the "Six dimensions of software project risk" by Wallace et al. (2004), requirements risk categories and linking to resolution techniques first identified by Mathiassen et al. (2007) and later complemented by Tuunanen et al. (2015).

After presenting the theoretical framework, the idea of Continuous Requirements Risk Profiling presented by Tuunanen et al. (2015) is discussed on higher level and the order of applying the method will be presented to the interviewee. After introducing how the method works, it is discussed how the testing is going to be done in the case project based on the interview agenda and what kind of answers are expected in each part. Interview themes 2, 3 and 4 are focused on testing and evaluating the model, first by assessing and prioritizing case project risk profile using the method. After conducting a risk analysis for the case project using the method, the interviewees are asked about method completeness, accuracy, understandability, usefulness and feasibility in project use. These themes are closely linked to MIS Success Measures by DeLone & McLean (1992) and target to get answer if the interviewees think that the method is feasible for their purposes.

The full interview questions used in theme-centered interviews can be found as Appendix A of this study.

6 TESTING AND IMPROVING THE CONTINUOUS REQUIREMENTS RISK PROFILING METHOD

The research was conducted as an interpretive case study, where the method was tested with several agile projects from same case company. When testing the method in selected projects, semi-structured theme-centered interviews were used as the primary data collection method. The collected qualitative data was analyzed with thematic analysis, to provide a comprehensive description of industry professionals' opinions, views and experiences about testing the Continuous Requirements Risk Profiling method. The results from the research were reflected against the tested model, existing literature and other research on the same topic.

6.1 Conducting the theme-centered interviews

The theme-centered interviews were conducted during July and August 2017. The potential interviewees were first discovered within the company based on two criteria: the use of agile or DevOps methodology in their project, and role related to requirements elicitation, specification, management and/or project management. After the discovery, the potential interviewees were contacted directly by email. The interview invitation described the topic, motivation and purpose of the study, asking if they were interested in the topic and willing to participate. All nine contacted potential interviewees accepted the invitation. The interviewed professionals were currently working with at least one project that used some agile development model and were working in roles such as project manager, product manager, system architect, technical lead or customer representative. Background information of the interviewees is summarized in table 21.

Table 21 Summary of interviewed professionals and their background information.

Interviewee pseudonym	Role	Industry experience
Interviewee 1	Senior Project Manager	Over 20 years
Interviewee 2	Senior Project Manager	Around 15 years
Interviewee 3	Project Manager and Senior Developer	Over 15 years
Interviewee 4	Systems Architect	Over 10 years
Interviewee 5	Customer Representative	Over 10 years
Interviewee 6	Technical lead and Developer	Around 10 years
Interviewee 7	Project Manager and Developer	Over 10 years
Interviewee 8	Developer and Junior Systems Architect	Around 5 years
Interviewee 9	Project Manager and Technical Sales	Over 15 years

The projects that were used as examples in the interviews had Scrum as the working methodology, complemented with some other agile working practices such as Continuous Integration. One of the projects was partially customer led, but in all other projects, the project management was done in the case company. The interviewed professionals had industry experience from five to more than 20 years, majority of them more than 10 years. All of the interviews were held as a video conference. Most of the professionals were interviewed individually, but to two of the interviews the professionals wanted to participate together with a colleague working in the same project. We allowed this, as the interviewed professionals understandably had busy schedules and this kind of discussion was seen beneficial for the projects. The audio of the interviews was recorded upon the approval of interviewed professionals and transcribed for thematic analysis. In the cases when interview language was Finnish, the transcribed data was translated to English during the analysis and the translated interview extracts are presented in this report. For anonymity and confidentiality of the interviews, only the interviewee role and approximate industry experience is reported with the extracts included in the study. The extracts are used to represent the themes and phenomena related to the overall data set, not individual opinions or experiences with clients.

6.2 Thematic analysis for interview data

Thematic analysis was used for analyzing the data collected with theme-centered interviews. The previously presented more structured, six-step approach described by Braun & Clarke (2006) was applied. Taking into account the nature and settings of this study as master's thesis, no specific text analysis program was used for coding and creating the thematic map. The data was coded using spreadsheets and thematic map with a free web-based diagramming tool draw.io.

In the first step, the interviews were listened, transcribed, read and checked for correspondence of the recordings. In the second step, initial codes were applied to the transcribed interview data and the data related to same codes was collated. A code was used to represents the meanings of certain subset of data, for example "Feedback on method checklist vocabulary". In the third step, candidate themes were searched from the coded data. A theme was used to represent the shared meanings of related codes, and identified candidate themes were for example "Managing requirements in agile software projects" and "Factors affecting to the completeness, understandability and accuracy of the method". As the interviews were conducted as theme-centered interviews, some of the themes were closely related to the interview themes, as Hirsjärvi & Hurme (2000) suggested. The discovered themes were also closely related to our research questions: the feasibility of the method, and how the method could be improved from the point of view of professionals working in industry. In addition, a rich collection of examples of identified, mitigated or realized requirements risks was extracted from the interview data for the use of the case company.

In fourth step, the found candidate themes were reviewed for representativeness for both the included codes and full data set, and possible overlapping between themes were considered. At the end of this step, a thematic map illustrating themes was created. This thematic map is presented in figure 6. In fifth step, the reviewed themes were revised, renamed and described. The sixth and final step was producing the reporting the results with the justification and interview extracts vividly describing each theme. The report is presented in next chapter.

6.2.1 Coding applied to dataset and identified themes

Table 22 Coding applied to dataset and identified themes.

Theme	Sub-themes and related coding
Theme 1: Managing risk in agile software projects	Subtheme: Context Code: Interviewee background general description Code: Interviewee role Code: Interviewee experience Code: Project background Code: Project type Code: Project description Code: Example of using the checklists Code: Use of requirements techniques
Theme 2: Lessons learned on requirements risk	Code: Identified risk Code: Example of identified risk Code: Example of relative impact of a risk Code: Example on resolving requirements risk
Theme 3: Factors affecting to method completeness, accuracy and understandability	Code: Feedback on checklist Code: Improvement/ change to checklists Code: Change to vocabulary Code: Addition to checklists Code: Relative impact in risk profiling table too high Code: Relative impact in risk profiling table too low Code: Feedback on risk profiling table presentation Code: Feedback on method completeness Code: Feedback on method accuracy Code: Feedback on method understandability Code: Open feedback related to method features
Theme 4: Factors affecting to method usefulness and feasibility in project use	Code: Opinion on usefulness of measuring relative impact Code: Feedback on risk resolution pattern Code: Suggested order Code: Example on resolving the risk pattern Code: Feedback on techniques Code: Improvement / change to risk resolution techniques Code: Used risk resolution techniques Code: Additions to resolution techniques Code: Feedback on method usefulness in agile projects Code: Feedback on method feasibility in agile projects

6.2.2 Thematic map of interview data

Based on the thematic analysis on interview data, four main themes were formulated to describe the patterns found in the interviews. The main themes discussed were named as "Managing risk in agile software projects", "Examples of requirements risk", "Factors affecting to method completeness, accuracy and understandability" and "Factors affecting to method usefulness and feasibility in project use". Each main theme contained several subthemes that represented the recurring patterns in related discussion.

For example, the discussion on "Theme 1: Managing requirements risk in agile software projects" contained subthemes describing the iterative approach to the activities, dependence of project characteristics, justifying the importance of requirements, different perceptions on responsibility and roles in risk analysis, and factors related to communicating about requirements risk management topic with customers. "Theme 2: Examples on requirements risk in agile software projects" contained subthemes describing discussion about identified requirements risk, typical realized requirements risk in agile software projects, successfully mitigated requirements risk and discussion about some risk items that were seen controversial. In the controversial risk items subtheme was debate whether the items were actual risks or just inherent qualities of software, which need to be accepted on reasonable level.

Themes 3 and 4 were closely related to the corresponding interview themes. Theme 3, "Factors affecting to method completeness, accuracy and understandability" described the discussion related to assessment of the method completeness, accuracy and understandability. This theme included several proposals what could be added to the method, how to improve the method so that it could provide more accurate interpretations of the situations faced in industry and how to make the method more understandable in terms of language and presentation.

Theme 4, "Factors affecting to method usefulness and feasibility in agile software projects" described the assessment and discussion of the method usefulness and feasibility

in case company's agile software projects. Subthemes described for example which features in the method are useful and which not, and would the method be feasible in the case projects and why, and the interviewed professionals' motivation to use the method in their future projects. The final thematic map is presented in figure 6 and the more detailed analysis of each thee is presented in next chapter.

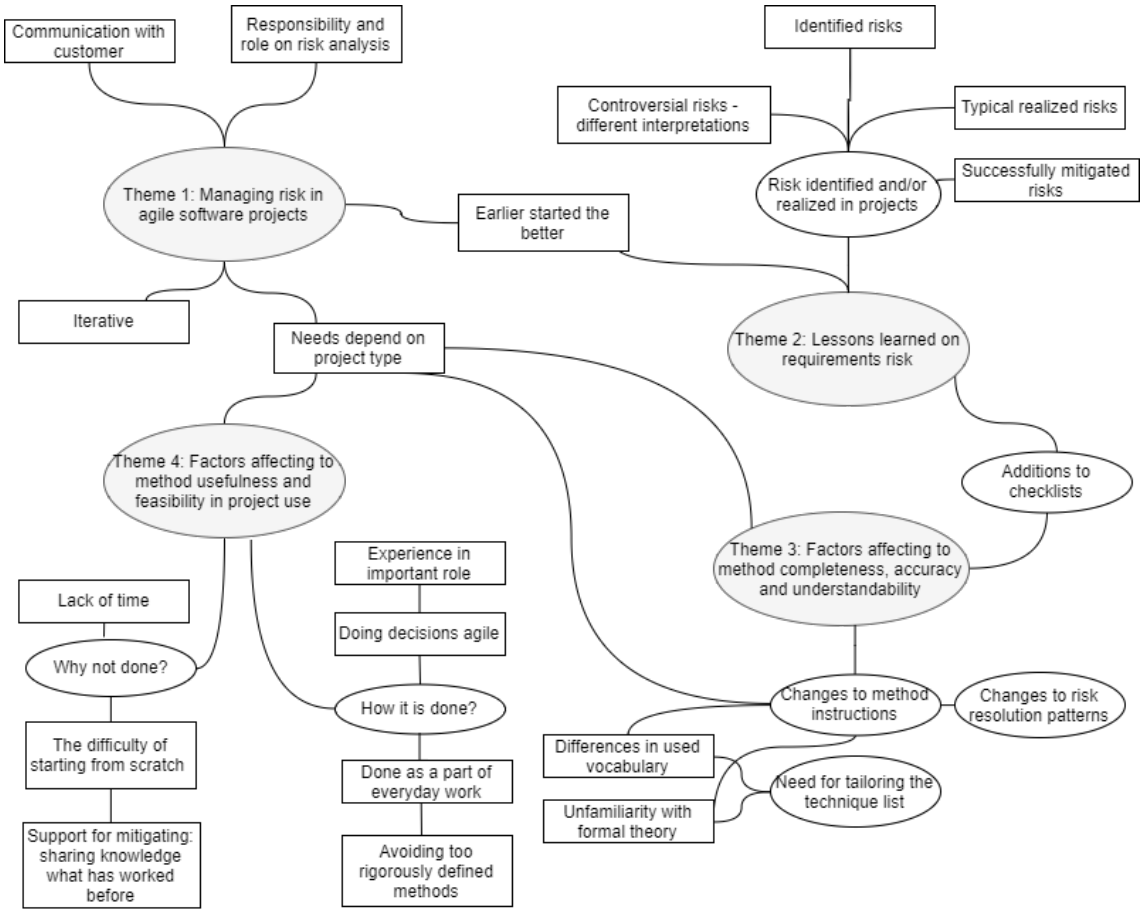

Figure 6 The final thematic map illustrating conducted thematic analysis on interview data, reviewed for correspondence and consistency of themes.

6.3 Analyzing the method feasibility in agile software development projects

This subchapter presents the analysis of the collected data set describing the results of testing a Continuous Requirements Risk Profiling method in interviews with industry professionals working in agile software projects. Overall, the interviewed professionals thought that identifying and managing requirements related risk was an important topic.

There was a nearly perfect agreement between the interviewed professionals that identifying the requirement risk is meaningful for the project success, and they were willing and motivated to use at least some parts of the tested method in future. All of the interviewees also gave some improvement proposals to the vocabulary of the checklists and the way the questions used for identifying requirements risks were formed. Presenting an improved version of the method, improving the method usability and making the risk items consistent among the checklists, was the most important prerequisite for adopting the method to project use in the case company.

6.3.1 Theme 1: Managing requirements risk in agile software projects

One of the identified themes concerned how the professionals working with agile projects in industry viewed managing requirements risk, and what kind of related observations they had made. Topics related to this theme contained justification and explanations for the importance of requirements, describing the iterative approach to the activities, dependence of project characteristics, different perceptions on responsibility and roles in risk analysis, and factors related to communicating about requirements risk management topic with customers.

When starting the discussion about requirements related risk, many of the interviewees brought up the importance of requirements engineering in the whole software development process. They also pointed out, that sometimes it was hard to explain the importance of requirement specification to the customers. Many of the interviewees then suggested that the prioritized requirements risk profile could work also as a medium on communication with the customer. A simple enough way to visualize the risk related to some software project related decision, consciously or unconsciously made, was seen very welcome and possibly useful by the interviewees.

> *"The benefit of requirements engineering is hard to explain and reason to some customers, even though it the only initial step you could buy from any software company and then go with that product [requirements specification]*

> *to order the actual implementation quite okay from any other software company. That if the requirements specification is well-made, it is really valuable also onwards."*
>
> Interviewee, Senior System Architect

One source of requirement related problems and risks was described to be associated with customer's lack of understanding about the requirements and software development process, and risks related to it. The interviewees described that resolving many of the requirement risks needed also the customer's participation and were not something that they could resolve only by their actions, so bridging this knowledge gap would be an important factor for successful requirements risk management and mitigation. In addition, when we take into account the agile philosophy and values presented in *Agile Manifesto* (Beck et al. 2001), individuals and interactions should be favored over processes and tools.

Also related to the problems of explaining the importance of extensive enough requirements work, many interviewees pointed out that sometimes it is hard to get the customers understand that building software is also creative work, and a good piece of software fulfilling the users' needs cannot be delivered like industry products. This observation has theoretical background from Brooks (1986), who stated that *software construction is a creative process*. The observation also enforces the view that software is more a medium of knowledge than a product (Tiwana & Keil 2004: 74), and requirements engineering is a very essential part of successful construction process:

> *"A software project is on a border of industrial work and creative work. It is not a pure industrial project, and it is not so easy to automate the software making process, it needs the creativity as well. It's kind of a mix of art and industry, that's the challenge."*
>
> Interviewee, Project Manager with over 15 year's industry experience

Workflows related to agile software development were also discussed, and how the phases of software development might be implemented differently between the projects. One noteworthy observation concerned the way method divides the identifying of the risk into three different phases, even though in some agile approaches the requirements, design and implementation activities can be done little by little, or even in parallel – that

would be the for example in XP (Beck 1999). Three of the nine interviewees brought up that in their case projects the design and requirements phases were done in parallel, and not as separate phases. They felt that in those cases it would be most convenient to merge the requirements phase checklist and the design phase checklist into one, or otherwise tailor the method:

> *"The initial design and the architecture was in the beginning, and it was a like a skeleton for everything else."*
>
> Interviewee, System Architect and Customer Representative

When discussing about providing indicative impact levels for each of the risk item, there was some opinions that the listing of relative impacts could not be valid, but majority of the interviewed professional thought that it could give valuable support to the decision-making related to risk management. Overall, the individual interpretations of the situations and situation-based decision-making was highlighted by several interviewees, especially by the ones who worked in more technically oriented roles. One of the opinions against too formally defined guidance to risk impact levels used previous arguments:

> *"Somehow, I think that this kind of listing is just impossible, or that it [the relative impact of each risk item] depends so much of the project, client or even involved people. The first thought is that this can't work [in all cases]."*
>
> Interviewee, Technical Lead

The interviewees' opinions about providing guidance to requirements techniques divided mainly into two categories: some of the interviewees were against of using formally defined techniques, and some of the interviewees thought it was positive if some new techniques would be introduced. Those interviewees though that the requirements technique listing would be beneficial especially for less experienced project managers. The interviewed professional's whose opinions were against of the use the formally defined techniques justified their opinion by arguing that use of such formally defined techniques would add unnecessary work and factors to the requirements process:

> *"But of course, the list contains all kinds of thinking techniques, but I have to say that I've never been a great fan of those, that kind of specified thinking techniques. Based on my experience it is better if this kind of processes go*

with their own pace, as using this kind techniques just adds quite a lot of overhead to that."

Interviewee, Senior System Architect

"*I'm not that familiar with this theory, so it's hard to present only names. But yes, I know very many ways to do the requirements. What I usually do to discover the requirements, is that I have certain templates and tools that I use, and then I have the experience. ... And usually I try to make requirements specification [as early as possible]. When you specify the requirements to some document, it work also as a mean of communication to make sure that the client understands, and as a reference when starting to do changes*."

Interviewee, Senior Project Manager and Technical Sales

Justification for these opinions can be found for example from the agile values (Beck et al. 2001).

6.3.2 Theme 2: Experiences on requirements risk in agile software projects

Overall, the interviews gave strong empirical evidence that the requirement risk items presented in checklists are valid and seen as risks among the interviewed professionals. The most experienced project managers pointed out that they identified all checklist items as valid risks, and there was not a single risk they had not seen to realize during their industry career. In the interviews, professionals told various vivid and valuable examples of the requirements risks which had identified or which had realized in the case projects. It speaks also for the validity of the risk items in the checklists, as industry professionals often face such risks in their projects:

"I think we didn't say for any [of the risk item] that this isn't not a risk or would not belong here."

Interviewee, Senior Project Manager

Some of the presented requirements risk items provoked also conflicting opinions among the interviewees. One of such items was for example "*Absence of project sponsor*" as some of the interviewees argued it to be a showstopper risk, and some of the interviewees were not sure if the risk item is relevant at all and will it affect to the project. This was an

interesting observation, and one possible explanation could relate to the interviewee background and role in the project: interviewees with developer or architect role were less concerned about this risk item, as the interviewees with project manager or technical sales role thought this was a high risk. On the other hand, previous observation can be also related to some factors in the case company settings and environment.

> *"I think some of these risks aren't even high impact risks, but they are showstoppers. A risk can be high and you can mitigate it, but for example that missing project sponsor, it is a showstopper. You don't have anyone, who would like to pay for it."*
>
> Interviewee, Project Manager and Technical Sales

The showstopper risks usually belonged to the requirements identity category, which also argues for the importance of mitigating the requirements identity risks first.

Another controversial risk item was *"Ambiguous requirements"*, which was identified and usually successfully mitigated in nearly all of the case projects. The more experienced professionals viewed such requirements as something that cannot be avoided at least in the earlier phases of the project:

> *"Have you ever seen a software project that doesn't have [ambiguous requirements]? I would like to see that. Yes, we have had those."*
>
> Interviewee, Senior Project Manager and Technical Sales

When discussing about the possible impact the risk, the opinions were again divided into two categories. One part of the interviewees found it as a high risk, but the other part presented that certain level of ambiguity needs to be tolerated and it might be even beneficial for the system design, if ambiguous requirements are refined before development phase:

> *"Well, I think that most of the requirements are usually on high level. Because, of course, not everyone has the technical depth to know how to do that from the implementation point of view. And it's kind of our job, to figure it out."*
>
> Interviewee, System Architect and Customer Representative

Missing requirements were suggested to often have same kind of root cause as ambiguous requirements. Several interviewees stated that based on their experience, missing requirements are usually related to the assumptions that something is already self-evident. Thus, if the other party does not specific enough either technical or business domain knowledge, it is hard to realize that a requirement is missing. Still missing requirements were seen as problematic, but something that cannot be completely avoided either:

> *"There are always some requirements that are missing, but that is because you cannot exactly specify everything to the last detail. You cannot know everything up to the last detail in requirements, because then they stop being requirements and became a technical specification. And that is probably only reason, why they are missing. Because for some of the people some requirements are already technical, and for some of them they are not."*
>
> Interviewee, System Architect and Customer Representative

Risk related to missing requirements was identified especially in those cases, when the projects were related to developing new features to existing software systems. Several interviewees also suggested that this could be taken into account also as its own requirement risk item, "Legacy concerns":

> *"Yes, probably there was missing quite many [requirements]. Or there were quite many things the customer assumed that we know. Or that we should know, because here is this old system as a basis. And those are quite challenging [situations], when you start checking afterwards, that yes there has been this kind of requirements for the 20 years old system, why it is not implemented in this new one."*
>
> Interviewee, Developer and Junior System Architect

In line with the agile ideology presented by Beck et al. (2001), also the interviewed professionals viewed change and change on requirements as an inherent part of software development. However, it seemed that agile was not always understood by everyone, which could be problematic:

> *"Often the customer believes that after you have once written the requirement, you don't need to change it that much. But in reality, that is just the starting point."*
>
> Interviewee, System Architect

The interviewed professionals described that fixed budget and timelines with agile methodologies were a common but also unavoidable risk. Almost all of the interviewees pointed out, that it was a potential risk in their project at least in one of the phases. Especially if other requirements risks were high, the professionals argued that fixed budget and timeline was a major threat. For example, emerging requirements dependency was described as one of the factors that can later change the initial estimates:

> *"Yeah of course that happens. ... Things happened that when doing the estimation, we thought that this is scope A and this is scope B. But when doing the actual design, we figured out that oops, we have to do a couple from the other as well."*
>
> Interviewee, System Architect and Customer Representative

The interviewed professionals argued that based on their experience, missing stakeholders were more common source of requirements risk than incorrect stakeholders. Several professionals suggested that missing stakeholders should be added to the checklists, and they justified their opinion with descriptions of cases where missing stakeholders had caused major changes in later phases of the project:

> *"It more the missing [stakeholders], which we have come across. For example, in one project we have changed almost everything as stakeholder from sales department joined [the project team] too late. And of course, it then affects to the costs, schedules and overall satisfaction."*
>
> Interviewee, Senior Project Manager

When discussing about lack of collaboration, several project managers described it to be "*a typical realized risk*". It was presented that sometimes despite the efforts to collaborate the team dynamics just do not work. Several reasons behind this typical problem were discussed and from the stories it was observed that besides requirements risk, this item is also interrelated to other Wallace et al. (2004)'s risk dimensions such as team risk or organizational environment risk:

> *"Maybe some people just get stuck. Usually it is one person doing some kind of design, and then tries to collaborate with others, but sometimes it just does not work. Some people just get closed inside a box for a week, and then they come out of the box and you find out that "Whoops, there are some problems" and then we need to get closed for a one week more."*
>
> Interviewee, Architect and Customer Representative

Unrated requirements were described to be a problem especially from the point of view of work estimates and project schedules. Related to unrated requirements, adding such risk items as "Unmeasurable requirements" and "Lack of context in requirements" was proposed. In some cases, unrated requirements could also lead to missing requirements:

> *"In requirements specification, it is said with one sentence that it shall produce the output of a certain calculation. The one sentence indicates that well it is only one calculation. But when we start to evaluate it in more detail it actually contains over 50 components, which properties we first need to calculate. And that means that we had really underestimated the needed work."*
>
> Interviewee, Senior Project Manager

Most of the interviewees thought that technology changes would be at least medium impact risk. One observation was that compared to project managers, developers assessed technology changes to have much higher impact to the project. Still, most of the interviewees viewed technology changes as something that:

> *"Technology changes are usually done so, that there is some backwards compatibility. That if something changes, it is not so radical. ... Often the technology changes to better, it does not change to worse. And it means that the client gets a better solution."*
>
> Interviewee, Senior Project Manager and Technical Sales

6.3.3 Theme 3: Assessment of the method completeness, accuracy and understandability

All of the interviewees gave some improvement proposals to the vocabulary of the checklists and the way the questions were formed. The most common and important from the perspective of the tool usability was making the risk items consistent among the checklists. In the method suggested by Tuunanen et al. (2016) some of the risk items were presented negatively and some positively. When testing the method with interviewees it was noticed that it is most convenient to present all checklist items as possible risks, so that answering "yes" means that a possible risk is identified:

> *"It is hard to formulate the answer... Because now this is not a yes or no question. Like do you have access to clients, then the answer would be yes, if missing access to clients then no, but it would be even more confusing"*
>
> Interviewee, Project Manager and Developer

In addition, different people understood some of the risk items differently. For example, "Delivering what the client requires" was suggested to be changed to "Unmet customer requirements" to clarify the situation.

Besides the previous improvement proposals, the method's checklists were found easy to learn and use, and many interviewees presented that checklists would be their preferred format to identify possible risks. There was a nearly perfect agreement among the interviewees that on the part identifying the requirements related risk method provided information that is complete, accurate and useful. Most of the interviewees presented that the method already had a comprehensive listing of possible requirements related risks, but also some additions to the checklists were presented. The less experienced industry professionals felt those useful for learning about possible requirement risks and adapting the overall idea. The more experienced industry professionals felt that the checklists were useful as those were seen as fast and efficient way to identify risks and plan further risk management.

Also analyzing the possible relative impacts with an improved version of the risk profiling table was found easy to learn and use with slight disagreement among couple of interviewees. On the contrary, almost all of the interviewees presented that intervening and resolving the risk with provided risk resolution pattern and tools would not be easy to learn and take into use. It was observed that the part of the method that suggests techniques to mitigate the risks was not as complete, accurate and useful as the other parts of the method, and would still need development. The resolution technique table was described inaccessible, hard to understand and overloaded with unexplained information:

> *"This list is way too long, it is unusable."*
>
> Interviewee, Customer Representative

There was a nearly perfect agreement among the interviewees that the requirements risks identified by the method were meaningful when thinking about the overall project success. Some of the interviewees had a strong opinion, that it should be necessary to think about the presented requirement risk items in every project. The importance of identifying the risks as early as possible was also highlighted. The most experienced project managers also pointed out, that if these risks are not identified, those cannot be mitigated either. Generally, they had the experience that once the risk is identified and communicated to the customer, also the customer is committed to resolve the risk for the good of both parties.

The interviewees agreed that the requirement risks presented in the method checklists were such risks that they had identified also with current risk management practices, but most of the interviewees felt that the risks were specified and categorized in the tested method. The most experienced project managers argued that they had actually seen all of the listed risks realize in a way or another, and presented that identifying those is very important. The checklists were also seen as a good way to identify and categorize requirement risk. The interviewees, who had less experience from risk management, felt that it was hard to identify all these risks without checklists, if one has not already experienced each risk realizing in some previous project:

> *"When you read these [risk items] from a checklist, you probably recognize things in different way as you do not know yet how to make it based on experience."*
>
> Interviewee, Developer and Junior Systems Architect

When discussing about the requirements risks that had been already identified in the examined projects, most of the interviewees presented that the method itself did not introduce such risks that they had not identified in the case project. However, several interviewed professionals also mentioned that previous was most probably due the fact that almost all of the case projects were already in implementation phase. The interviewees hypothesized, that if the method had been applied already in the requirements or design

phase, most probably there would have been some new risks identified. Several interviewees had also faced situations where they had identified some requirement risk, but felt that the risk was out of the scope they could affect and resolve.

Based on the overall results related to requirements techniques it was observed that most of the techniques were unfamiliar to the interviewees. In general, each interviewee was familiar on average six to eight techniques of the total 85 techniques presented in the table. The requirements techniques that at least one interviewee mentioned to be familiar with were brainstorming, card sorting, data-flow diagrams, focus groups, prototyping, testing, workshops, mockups, use cases, interviews, state charts, business process analysis, quality function deployment, user interface prototyping, laddering interviews and deriving requirements from existing system. Besides the formal techniques, interviewees mentioned to use practices such as maintaining the statuses of requirements, finding out the background of each requirement, common sense and asking the question why.

> *"I am not that familiar with this theory, so it is hard to present only names. But yes, I know very many ways to do the requirements. ... I have certain templates and tools that I use, and then I have the experience. ... When you specify the requirements to some document, it work also as a mean of communication to make sure that the client understands, and as a reference when starting to do changes."*
>
> Interviewee, Senior Project Manager and Technical Sales

It should be noted that the unfamiliarity with the technique names and their theoretical background might have affected to the interview results. As most of the techniques were unknown, those would have needed some explanation before the interviewees could even evaluate if the technique was feasible in their project and unfortunately, the interview schedules did not allow detailed discussions on each of the techniques. There were also several comments, that the interviewees were not aware that all the techniques presented in the table were such techniques that could be used on their own. The current presentation in requirements risk resolution techniques table was described as too long, unusable, frustrating and overloaded with unexplained information. These reactions could be explained by the previously presented factors related to overall attitudes towards formally defined requirements techniques and unfamiliarity with most of the listed techniques.

To make the risk resolution table feasible in the case company, a new tailored requirements technique toolbox will be proposed. The core set of included techniques will be selected from the techniques that the industry professionals mentioned in the interviews and had already found feasible in their projects. The already familiar techniques will be complemented with some new techniques that could be feasible in the case company. The techniques included in the toolbox should be also introduced shortly, so that the industry professionals could easily take the techniques into use.

6.3.4 Theme 4: Assessment of the method usefulness and feasibility

Several interviewees argued that often the risks related to requirements originate from false presumptions, that the customer is familiar with software development process and the developer is familiar with the customer's operations and environment. In addition, many of the items on the checklists were seen to reflect this. When discussing about the presumption, one of the interviewed senior project managers emphasized the importance of requirements risk management discussion and getting rid of self-evidences:

> *"We have to get rid of the idea that some things are self-evident. Those must not be seen as self-evident."*
>
> Interviewee, Project Manager

There was a substantial agreement among the interviewees that the tested method itself did not bring anything completely new to the requirements risk management in agile software project, but the checklists were seen as a useful tool for identifying possible requirements risks. In addition, the learning value and the information provided by the method was described as "*thought provoking*" for both experienced and new project managers. There was a nearly perfect agreement between the interviewees that the checklists could be easily applicable to the workflow of those agile software projects they were currently working in. The interviewed professionals described that they would use the method as project management tool for identifying risk, but also as a medium of communication to provoke discussion with the project team and the client participating to the requirements work.

However, the interviewed professionals were not sure if the method would provide enough support for planning the risk management actions in real situations. The interviewees pointed out that the method itself did not provide complete and clear enough suggestions for risk resolution actions. Thus, this one of the parts in method that would still need development. The risk resolution pattern was received well among the interviewees, but the list of linked requirements techniques for resolving each type of requirements risk got more critique. There was a moderate agreement among the interviewees that the proposed risk resolution pattern was accurate, and all interviewees agreed that identity risks are the ones that need to be resolved first. The interviewed professionals who disagreed with the proposed risk resolution pattern presented that after resolving the identity risks the next three requirements risk types cannot be put in order, the order would need to be changed or that the order depends of some other project characteristics:

> *"I think the steps two, three and four... You cannot put them into just one order. ... Depending of the project, I would discuss with the stakeholder or client which one we should resolve first."*
>
> Interviewee, Project Manager and Technical Sales

Most of the interviewees could not provide opinions if using the method could improve the project level decision-making and overall outputs, which was expected already when designing the study. There was uncertainty in the answers as the actual impacts could not be predicted, but the interviewed professionals presented that some improvement could be possible. One opinion was that it could improve the overall requirements process, if both the development team and customer would commit in using this method and reviewing the risk reports before continuing to next project phase. Some interviewees also presented that identifying the requirements related risks earlier could have positive impact on the overall project success. They based their arguments on the experiences from previous projects, and presented that identifying the risks and discussing of those both internally within the project team and externally with the customer often had positive impact to the overall success. The same interviewees also had experiences that the customer had improved their way of working based on the risk analysis feedback.

Overall, all of the nine interviewed professionals presented that they would like to use some parts of the Continuous Requirements Risk Profiling method in future. There were several suggestions how the method could be taken into use in the case company. Interviewed industry professionals presented that the method should not be too strictly and formally defined so that it could be tailored to fit as many projects as possible, but the industry professionals hoped that more corresponding tools would be available for use:

> *"I would say that this all is very useful when thinking about our projects. ... We do have the knowledge, but it would be good to also concretize and maintain it."*
>
> Interviewee, Project Manager

Still, there was conflicting opinions should risk analysis be compulsory or voluntary. Around half of the interviewees thought that it could be determined in the quality manual that projects need to do risk analysis. At the same time, other half of the interviewees preferred that the use of risk analysis should be rather voluntarily than forced, depending of each project's individual situation. Several interviewees also suggested that when applying the method, the project team and customer could be involved in the risk management and informed about the results. One of the suggestions was creating a small printable leaflet of the method to make it always visible and accessible. This kind of approach could fit also to the agile working practices of the projects, and would not add unnecessary processes to project work. Another suggested agile approach to continuous requirements risk profiling was "checklist without checks":

> *"One thing that I like about checklists, is that they give you an idea what you should be thinking about in requirements. The problem with the checklists is though, that you are forced to fill them. So, I would skip that part of actually filling in the checklists."*
>
> Interviewee, Customer Representative

6.4 Proposing improved Continuous Requirements Risk Profiling method

Based on the results of this study, also an improved and tailored version of Continuous Requirement Risk Profiling method was suggested for the use of the interviewed professionals working in case company. The improvements were based on industry professionals' needs and were chosen among all suggestions applying following criteria:

1) Strong empirical justification: More than one of the interviewees request the same improvement to the method.

2) Or empirical justification with theoretical justification: Evidence could be found from the existing literature that the single observation is valid.

6.4.1 Tailoring method instructions to case company environment

Based on the interviewee results, the industry professionals were willing to adopt the method, but did not want it to be too formally defined how the method should be used. Thus, the instructions for conducting risk analysis with method were tailored to take into account agile environment and the ways of working in case company. The tailored method instructions are presented in table 23. The original method instructions according to Tuunanen et al. (2016) were presented in table 13.

Table 23 Conducting risk analysis using the Continuous Requirements Risk Profiling method and applying the risk resolution pattern adapted from Tuunanen et al. (2016) tailored to case company context.

Step 1: Identify risks with checklists in each requirements, design and implementation phase. If some of the phases are parallel or overlapping, merge the checklists.

Step 2: Assess project risk profile by considering individual requirements risk factors possibly affecting the project. The indicative impact levels can be used as a guideline for prioritizing requirements risks, but pay attention also to project characteristics that might affect to the indicate impact levels.

Step 3: Intervene with Requirements Risk Resolution Techniques according to the risk resolution pattern. Suggestions of some common requirements techniques are presented in the toolbox.

1. If identity risks are high, put high emphasis on discovery techniques.
2. If integrity risks are high, put high emphasis on prioritization techniques.
3. If volatility risks are high, put high emphasis on experimentation and specification techniques.
4. If complexity risks are high, put high emphasis on specification and experimentation techniques.

After resolving possible identity risks, it is suggested to consider if some of the following risks (2 to 4) has significantly higher relative impact to the project. If so, apply the pattern by resolving the highest risk first.

6.4.2 Additions and changes to checklist risk items

The interviewed professionals also suggested several additions to the method checklists. As presented also in the previous chapters, majority of the interviewed professionals stated that they were not familiar with the theoretical framework and the definition of requirements risk. Thus, besides the main selection criteria, the suggestions were also

checked for correspondence to definition of the requirements risk by Wallace et al. (2004). All of the suggested additions checklists are presented in table 24.

Table 24 Additions suggested to requirements risk checklists.

Legacy concerns not taken into account in requirements
Essential requirements not formulated or agreed before starting the project
Architect (or technical) support not available for requirements work
Lack of understanding of software project dynamics and process
Lack of context on requirements
Unmeasurable requirements
Requirements understood differently among project team and stakeholders
Changes in requirements not taken into account in budget and timelines
Lack of third party commitment and co-operation
Customer lacking resources for requirements work
Changes on the source of requirements
Unclear timelines
Project team members experience not taken into account in work estimates

Some of the suggested additions were not actual requirement risks, as they directly related more to team dynamics or project management. The additions that belonged to the requirements risk category were Legacy concerns not taken into account in requirements, Architect support not available for requirements work, Lack of context on requirements, Unmeasurable requirements, Lack of third-party commitment and co-operation and Changes on the source of requirements. The interviewees were satisfied with the overall design of the checklists, but some changes were requested to the vocabulary used in of the checklists. The changes concerned some individual terms and the way in which some risk items were positively and others negatively formed. The checklists with previous additions and changes are presented below in tables 25, 26 and 27, added and changed items highlighted with **bold** notation.

Table 25 Requirements phase checklist initially presented by Tuunanen et al. (2016) and improved by feedback collected in this study, additions and changes **bolded**.

Risk	Risk Type	Project is/can be exposed?
Project **Financing Unknown or** Missing	*Identity*	
No Access to Clients (**Distance to Original** Source of Requirements)	*Complexity*	
Ambiguous Requirements	*Identity*	
Architect or Other Technical Support Unavailable	*Complexity*	
Change in Client's Business Strategy and Direction	*Volatility*	
Change in External Regulations	*Volatility*	
Change in the Source of Requirements	***Identity***	
Missing Client Commitment	*Identity*	
Constrained Users' Knowledge	*Complexity*	
Fixed Budget and Timelines	*Integrity*	
Incorrect or Missing Stakeholders	*Identity*	
Lack of Context in Requirements	***Identity***	
Legacy concerns	***Integrity***	
Misunderstood Business Needs	*Identity*	
Underestimation of Change Magnitude	*Volatility*	
Unmeasurable Requirements	***Identity***	
Unrated Requirements	*Volatility*	
Any other risks that could affect design and implementation		

Table 26 Design phase checklist initially presented by Tuunanen et al. (2016) and improved by feedback collected in this study, additions and changes **bolded**.

Risk	Risk Type	Project is/can be exposed?
Ambiguous Requirements	*Identity*	
Change in External Regulations	*Volatility*	
Missing Client Commitment	*Identity*	
Conflict with External Regulations	*Identity*	
Conflicting Requirements	*Integrity*	

Risk	Risk Type	Project is/can be exposed?
Missing Requirements	*Identity*	
Unmet Customer Requirements (Unable to Deliver What the Client Requires**)**	*Identity*	
Emerging Requirements Dependency	*Volatility*	
Fixed Budget and Timelines	*Integrity*	
Knowledge Gap between Coworkers	*Complexity*	
Lack of Collaboration	*Complexity*	
Lack of Third-party Commitment and Co-operation	***Integrity***	
Legacy Concerns	***Integrity***	
Technology Changes	*Volatility*	
Underestimation of Change Magnitude	*Volatility*	
Unrated Requirements	*Volatility*	
Unmeasurable Requirements	***Identity***	
Any unresolved risks from requirements and risks that could affect implementation		

Table 27 Implementation phase checklist initially presented by Tuunanen et al. (2016) and improved by feedback collected in this study, additions and changes **bolded**.

Risk	Risk Type	Project is/can be exposed?
Ambiguous Requirements	*Identity*	
Change in External Regulations	*Volatility*	
Missing Client Commitment	*Identity*	
Fixed Budget and Timelines	*Integrity*	
Hostile Users	*Identity*	
Lack of third-party commitment and co-operation	*Integrity*	
Legacy Concerns	*Integrity*	
Project Team Member **Changes**	*Volatility*	
Underestimation of Change Magnitude	*Volatility*	
Unrated Requirements	*Volatility*	
Unmeasurable requirements	***Identity***	
Any unresolved risk items from design and requirements		

6.4.3 Improvements to risk profiling table

Improvements suggested to risk profiling tables were related to both presentation and the estimated relative impacts of each risk item. Different ways to present the information with less duplication were introduced. The most often suggested were either splitting the risk profiling table into three parts, corresponding to the checklists, or presenting the risk profiling table in four parts, where the fourth table would present the risks affecting to all phases. Some of the interviewees also suggested that the risks affecting to all phases most probably did not have same impact on all phases, thus it was observed that there was already less duplication as the impact depended of the phase. The changes suggested to presented relative impacts of each risk item are listed in table 28.

Table 28 The changes suggested to presented relative impacts of risk items.

Risk item	Original impact	Suggested impact
Ambiguous requirements	High (all)	**Medium** (requirements, design), **High** (implementation)
Constrained by user's knowledge	Low	**Medium**
Fixed budget and timelines	Medium	**High**
Hostile users	Medium	**Low**
Knowledge gap between coworkers	Medium	**High**
Technology changes	Low	**Medium**
Underestimation of change magnitude	Medium (all)	**Medium** (requirements, design), **High** (implementation)

The changes concerned the indicative impacts of risk items Ambiguous requirements, Constrained by user's knowledge, Fixed budget and timelines, Knowledge gap between coworkers, Technology changes and Underestimation of change magnitude. The risk profiling tables with following changes are presented below in table 29, added and changed items highlighted with **bold** notation.

Table 29 Indicative risk impact levels initially presented by Tuunanen et al. (2016) and improved by feedback collected in this study, additions and changes **bolded**.

Requirements Phase Specific Risks	Impact	Design Phase Specific Risks	Impact	Implementation Phase Specific Risks	Impact
Incorrect or Missing Stakeholders	High	Missing requirements	High	Hostile users	**Low**
Misunderstood business needs	High	Knowledge gap between coworkers	**High**	Lack of third-party commitment and co-operation	Medium
No access to clients	High	Unmet Customer Requirements	High		
Project Financing Unknown or Missing	High	Conflict with external regulations	Medium		
Architect or Other Technical Support Unavailable	Medium	Conflicting requirements	Medium		
Change in Client's Business Strategy and Direction	Medium	Emerging requirements dependency	Medium	Project Team Member Changes	Medium
Constrained by users' knowledge	Medium	Lack of third-party commitment and co-operation	Medium		
		Lack of collaboration	Medium		
		Technology changes	**Medium**		
Risks Affecting All Phases					
Ambiguous requirements	**Medium**	Ambiguous requirements	**Medium**	Ambiguous Requirements	High
Fixed budget and timelines	**High**	Fixed budget and time lines	**High**	Fixed budget and timelines	**High**
Missing client commitment	High	Missing client commitment	High	Missing client commitment	High
Unrated requirements	High	Unrated requirements	High	Unrated requirements	High
Change in external regulations	Medium	Change in external regulations	Medium	Change in external regulations	Medium
Legacy concerns	**Medium**	Legacy concerns	**Medium**	Legacy concerns	**Medium**
Underestimation of change magnitude	Medium	Underestimation of change magnitude	Medium	Underestimation of change magnitude	**High**
Unmeasurable requirements	**Medium**	Unmeasurable requirements	**Medium**	Unmeasurable requirements	**Medium**

6.4.4 Introducing the tailored requirements technique toolbox

As Mathiassen and Tuunanen also suggested in their article "*Managing Requirements Risk in IT Projects*" (2011), organizing a balanced toolbox that fits the organizations needs is the final preparation step when starting the requirements risk management. Based on the empirical data collected from the case company industry professionals, we organized the initial toolbox from techniques that were already familiar and in use within the company. The selection criterion for the techniques included in the toolbox was, that at least one of the interviewees was previously familiar with the technique. The used techniques were distributed quite evenly between categories. Table 30 presents the tailored requirements techniques toolbox, modified from full listing which Tuunanen et al. (2016) had originally adapted from Mathiassen et al. (2007). The original, full list by Mathiassen et al. (2007) can be found as Appendix B of this study.

Table 30 Requirements risk resolution technique list and categorization, adapted from Mathiassen et al. (2007: 594–596) and filtered based on the results to specify the most well-known techniques among the interviewed professionals.

Name	Specification	Experimentation	Discovery	Prioritization
Brainstorming			*	
Card sorting			*	*
Contextual design	*		*	*
Cooperative prototyping		*		
Defining critical success factors			*	*
Data flow diagram	*			
Deriving requirements from existing system			*	
Entity-relationship modeling	*			
Focus group			*	
Laddering Interviews			*	
Open interview			*	
Participatory design		*	*	

Process analysis			*	
Prototyping		*		
Quality function deployment	*			*
Requirements prototyping		*		
Requirements workshops			*	
State charts	*			
Structured walkthroughs			*	
Testing		*	*	
Use cases			*	
User group			*	
User-interface prototyping		*	*	

7 DISCUSSION

In this study, a software project risk management tool Continuous Requirements Risk Profiling method was examined, tested and improved employing the input and experience of industry professionals working in agile software projects. The research answered two research calls. First, the call by Tuunanen et al. (2015: 4026–4027) to assess the feasibility of proposed Continuous Requirements Risk Profiling method in real life agile software projects. Second, the request by Mathiassen et al. (2007: 583) to assess the usefulness of presented requirements engineering techniques in different contexts and use cases, now as a part of tested Continuous Requirements Risk Profiling method. The main research questions for which this study sought answers were:

1. Does the developed continuous requirements risk profiling method fit the needs of agile software development projects?
2. Would the developed continuous requirements risk profiling method help industry professionals to identify such project characteristics that are seen as risks for the project success?
3. How the continuous requirements risk profiling method should be improved, so that the industry professionals would find it easy to use and useful in their every-day work?

The answers for previous research questions were sought by conducting a case study and collecting data with theme-centered interviews with professionals working in industry. With each interviewee, the method was applied to one case project that the interviewee could choose freely. The requirements for the project were that it used some kind of agile methodology and the interviewee was participating in the requirements work in that specific project. The collected qualitative data was analyzed with thematic analysis, resulting four main themes across the data set. The conducted study provided answers to all of the previous research questions.

For the question if the developed Continuous Requirements Risk Profiling method fit the needs of agile software development projects, the results of the study indicated that at

least in the case company tested method did fit the needs of agile software development for identifying requirements related risks. Industry professionals found that the method would fit well to the workflows of agile projects and most parts of the method were perceived as easy to learn and use. Still after the ideology and values behind agile software development approaches (Beck 1999, Beck et al. 2001, Poppendieck & Cusumano 2012), the professionals did not want to adopt too heavy or rigorously defined methods.

When thinking about answers to the questions that would the developed continuous requirements risk profiling method help industry professionals to identify such project characteristics that are seen as risks for the project success, the answers were promising at least in the case company context. The continuous requirements risk profiling method helped the interviewed professionals to identify several project characteristics that they agreed to be risks for the project success. The characteristics discussed in the method proposed by Tuunanen et al. (2016) could have posed a serious threat for project success if not mitigated and had often needed managerial intervention. On the other hand, some of the characteristics method suggested as possible requirements risks were seen as integral and unavoidable part of software development (Brooks 1987, Ramesh et al. 2010) that cannot be avoided at least in rapidly changing environments and agile software development.

For the third question about how the continuous requirements risk profiling method should be improved that the industry professionals would find it easy to use and useful in their everyday work this study provided many proposals. Still, the answers to this question are probably more opinionated and tied to the case company environment. One improved and tailored version of the continuous requirements risk profiling method was proposed as a part of this study. At least in the case company environment, the industry professionals favored a less strictly defined version of the method that could be easily adapted to different kind of projects and situations. This follows also the initial idea that Mathiassen et al. (2007: 570) had when building the theoretical framework: the synthetized theoretical framework should be simple, yet comprehensive enough to understand the possible requirements development risks and techniques. The checklists were found easy to use and useful for identifying the risks, and only some additions to risk items were

requested. The risk profiling table and requirements techniques listing needed still more development, and the interviewed professionals requested changes both to the contents and presentation. It seemed that for industry and agile software development context, more simplified presentation would be preferred.

The main contributions to theory of this study were to provide validation for the existing theory in practice, provide improvement proposals to the theory based method from practice and industry professionals, and provide a set of examples how industry professional view and manage requirements related risk in agile software projects. The main contribution to practice was introducing the theory and method to industry professionals within the case company, and providing an empirically validated and improved version of the method to the use of professionals working in the case company. As a main limitation, it should be noted that the study was conducted as an interpretive case study. Thus, the results are related to the environment and organizational setting of the case company and might not apply directly to other contexts or to all agile software projects.

8 CONCLUSIONS

The results of this study show that identifying and managing requirements risk is still an important topic in the field of software engineering, and the industry professionals are motivated to adopt new methods and knowledge to better understand and identify requirements risk. The theoretical background for such methods exists, but the theory and suggested methods are not itself familiar for the professionals working in industry. The results show that managing requirements risk in industry is often driven mainly by previous professional experience. However, also the interviewed professionals point out that this can lead to incomplete interpretations of the risks and related phenomena.

Based on the results the tested Continuous Requirements Prioritization method was feasible with small improvements for identifying requirements related risk in the agile software development projects examined in case company. The interviewed professionals working in industry found the method useful for detecting and prioritizing several kinds of requirements management related risks, and the checklists provided by the method contained a complete and accurate enough overview of possible requirement related risks. The interviewed professionals found the model useful especially for less experienced project managers. Still even the more experienced project managers found the model thought provoking and a useful tool in the rapidly changing environment they were working in. All of the interviewed industry professionals working in agile software development projects agreed that the presented model was feasible with the current working practices and applicable to the workflow.

The interviewed professionals suggested improvements to the method regarding the used vocabulary, additions to the checklists, changes to the presentation and indicative impact levels presented in the risk profiling table. Also based on the results it was clear, that the overall risk resolution suggestions should be designed more carefully to be beneficial for the industry professionals working with agile software development. There was an agreement between the interviewed professionals regarding most of the suggested improvements.

For practice and professionals working in industry, it is expected that already providing such methods and knowledge to industry professionals will already help project managers to pay more attention to requirements related risks. The results suggest that the tested method can provide support for identifying and prioritizing the risks on high level, but the more precise prioritization and finding resolutions to the risks was seen to require also practical experience and consideration of the project specific characteristics. The results prove that the model could be taken into use in case company with previously presented modifications, and the model would be feasible in majority of agile software projects run in case company. Based on the observations made from the interview results, it was clear that it would be beneficial to make everyone more aware of the available tools, and embodying the existing knowledge into methods or "lessons learned" knowledge is viewed valuable. The results suggested that in case of agile software projects, the learning value professionals working in industry could get from the method was seen more important than formally applying the method on defined intervals.

For researches, the results of this case study suggest that the overall results of the feasibility of the method are expected to be positive. Improvements suggested to the presentation, additions to checklists and further development of the resolutions are some suggestions that could make the method easier to adopt by industry professionals. Evaluating and testing of this improved model is suggested to make these findings more generalizable. The researchers can also notice from other similar studies that the needs of agile software projects might be different, and most probably one model will not suit all projects and teams.

REFERENCES

Ahmad, M., Markkula J. & Ovio M. (2013). Kanban in software development: A Systematic literature review. *2013 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, Sept. 2013, pp. 9–16.

Beck, K. (1999). Embracing change with extreme programming. *Computer*, 32(10), pp. 70–77.

Beck, K. & Beedle, M. & Bennekum, A. & Cockburn, A. & Cunningham, W. & Fowler, M. & Grenning, J. & Highsmith, J. & Hunt, A. & Jeffries, R. & Kern, J. & Marick, B. & Martin, R. C. & Mellor, S. & Schwaber, K. & Sutherland, J. & Thomas, D. (2001). Manifesto for Agile Software Development. [Web article]. *Agile Alliance*. Available: http://Agiloemanifesto.org. Accessed August 14th, 2017.

Boehm, B. (1976). Software Engineering. *Computers, IEEE Transactions on, C-25*(12), pp. 1226–1241.

Braun, V. & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology* (3:2), pp. 77–101.

Brooks, F. P. (1987). No Silver Bullet Essence and Accidents of Software Engineering. *Computer*, 20(4), pp. 10–19.

Cao, L. & Ramesh, B. (2007). Agile Software Development: Ad Hoc Practices or Sound Principles? *IT Pro*, 9(2) March / April, pp. 41–47.

Chen, L. 2015. Continuous Delivery: Huge Benefits, but Challenges Too. *IEEE Software* (32:2), pp. 50–54.

DeLone W. H. & McLean E. R. (1992). Information System Success: The Quest for the Dependent Variable. *Information System Research* 3:1, 60–95.

Ebert, C., G. Gallardo, J. Hernantes & N. Serrano (2016). DevOps. *IEEE Software*, 33(3), pp. 94–100.

Ebert, C., P. Abrahamsson & N. Oza (2012). Lean Software Development. *IEEE Software*, 29(5), pp. 22–25.

Fowler, M. (2006). Continuous Integration. [Web article]. Available: http://martinfowler.com/articles/continuousIntegration.html. Accessed January 9th, 2017.

Hickey, A. M. & Davis, A. (2004). A Unified Model of Requirements Elicitation. *Journal of Management Information Systems*, (20:4), pp. 65–84.

Hirsjärvi, S. & Hurme, H. (2000). *Tutkimushaastattelu: Teemahaastattelun teoria ja käytäntö*. Helsinki: Helsinki University Press. ISBN: 951-570-458-8. 213 p.

Iversen, J. H., L. Mathiassen & P. A: Nielsen (2004). Managing Risk in Software Process Improvement: An Action Research Approach. *MIS Quarterly* (28)3, pp. 395–433.

Kauppalehti (2017). Lääkesotkusta voi tulla Oriolalle pitkä lasku – päivittäin yli sata hätätilausta. [Web article.] *Alma Media.* [Accessed January 20th, 2018.] Available: https://www.kauppalehti.fi/uutiset/laakesotkusta-voi-tulla-oriolalle-pitka-lasku--paivittain-yli-sata-hatatilausta/WdHMTi65?_ga=2.214112404.287771892.1518954981-1966970483.1518954981.

Mathiassen, L., Saarinen, T., Tuunanen, T. & Rossi, M. (2007). A Contingency Model for Requirements Development. *Journal of Association of Information Systems*, (8:11) November, pp. 569–597.

Mathiassen, L., Saarinen, T., Tuunanen, T. & Rossi, M. (2008). *Managing Requirements Engineering Risks: an Analysis and Synthesis of the Literature.* All Sprouts Content. 82 p.

Mathiassen, L. & Tuunanen, T. (2011). Managing Requirements Risks in IT Projects. *IT Professional*, 13(6), pp. 40–47.

Napier, N., Mathiassen, L. & Johnson, R.D (2009). Combining Perceptions and Prescriptions in Requirements Engineering Process Assessment: An Industrial Case Study. *IEEE Transactions on Software Engineering*, 35(5), pp. 593–606.

Persson, J. S., Mathiassen, L., Boeg, J., Madsen, T. S., & Steinson, F. (2009). Managing risks in distributed software projects: an integrative framework. *IEEE Transactions on Engineering Management*, (56: 3), pp. 508–532.

Poppendieck, M & Cusumano M. A. (2012). Lean Software Development: A Tutorial. *IEEE Software*, 29(5), pp. 26–32.

Ramesh, B., L. Cao & R. Baskerville (2010). Agile requirements engineering practices and challenges: An empirical study. *Information Systems Journal*, 20(5), pp. 449–480.

Rinko-Gay, B. (2013). You May Be a Scrum-But. [Web article]. *Scrum Alliance*. [Accessed 16.7.2017]. Available: https://www.scrumalliance.org/community/articles/2013/february/you-may-be-a-scrum-but.

Saarinen, T., & Vepsäläinen, A. (1993). Managing the risks of information systems implementation. *European Journal of Information Systems*, (2:4), pp 283–295.

Schwaber, K. & Sutherland, J. (2016). The Definitive Guide to Scrum: The Rules of the Game. The Scrum Guide. [Web article]. *Scrum Alliance*. [Accessed 30.9.2017]. Available: https://www.scrumalliance.org/why-scrum/scrum-guide.

Tiwana, A. & Keil, M. (2004) The One Minute Risk Assessment Tool. *Communications of the ACM*, (47) 11, pp. 73 –77.

Tuunanen, T. (2003). A New Perspective on Requirements elicitation methods. *JITTA: Journal of Information Technology Theory and Application*, 5(3), pp. 45–62.

Tuunanen, T. & Kuo, I-T. (2015). The effect of culture on requirements: A value-based view of prioritization. *European Journal of Information Systems*, 24(3), pp. 295–313.

Tuunanen, T., T. Vartiainen, M. Ebrahim & M. Liang (2015). Continuous Requirements Risk Profiling in Information Systems Development. *48th Hawaii International Conference on System Sciences (HICSS)*, 2015, Jan. 2015, pp. 4019–4028.

Tuunanen T. (2016). Development of Requirements Risk Prioritization Method. Unpublished journal article. 29 p.

Wallace, L., Keil M. & Rai A. (2004). Understanding Software Project Risk: A Cluster Analysis. *Information & Management 42*, no. 1 (2004), pp. 115–125.

Walsham, G. (1995). Interpretive case studies in IS research: Nature and method. *European Journal of Information Systems, 4*(2), pp. 74–81.

APPENDIX A

Interview themes and questions

Introduction
Introducing the interviewer and research topic. Describing the interview practices and what is on the agenda: testing feasibility of the developed continuous requirements risk profiling and prioritization method in a case project.
Q: The interview will take around one to maximum one and a half hours, is it still a suitable schedule for you?
Q: Would you give your permission to record the interview for further analysis?
Q: Do you have any questions related to the interview practices?
Q: Please introduce yourself and tell little about your current role in this project

Next, the theoretical background is presented to the interviewee. When testing the interview questions with a case company representative, it was noticed that presenting the theoretical background for the interviewee would help to lead the conversation to right direction. Introduced theory is the "Six dimensions of software project risk" by Wallace et al. (2004), requirements risk categories and linking to resolution techniques first identified by Mathiassen et al. (2007) and later complemented by Tuunanen et al. (2015). After that the idea of continuous requirements risk profiling and management presented by Tuunanen et al. (2015) is discussed on higher level and the order of applying the method will be presented to the interviewee. After that it is discussed how the testing is going to be done in the case project based on the interview agenda and what kind of answers are expected in each part.

Introduction to the case project requirements
Requesting general introduction of the case project and its high-level user stories/use cases/requirements.
Q: Which user group(s) needs user stories/use cases/features are related?
Q: Which project stakeholder's work or operations are affected by the implementation of these user stories/use cases/features?
Q: Depending of the project size, do you want to test the model by assessing the whole project or certain one or two user stories/use cases/features from it?

Theme 1: Introducing the interviewee and the case project user stories/use cases/requirements

Requesting general introduction of the case project and its high-level user stories/use cases/requirements.
Q: Which user group(s) needs user stories/use cases/requirements are related?
Q: Which project stakeholders' work or operations are affected by the implementation of these user stories/use cases/requirements?
Q: Depending of the project size, do you want to test the model by assessing the whole project or certain one or two user stories/use cases/features from it?

Presented interview structure was:
Identifying the risks
- Requirements phase checklist
- Design phase checklist
- Implementation phase checklist

Assessing the risk profile
- Project risk profiling table

Prioritizing the risks and reviewing presented risk resolution patterns
- Risk resolution patterns

Reviewing the suggested risk resolution techniques
- Risk resolution techniques

Common wrap-up about applying the method

4.4. Theme 2: Assessing and prioritizing case project risk profile using the method

Identifying the risks
Each project phase (requirements, design and implementation) has its own checklist, which takes into account the impact of the risk driver in specific project phase.
Please go through each row from relevant checklists and assess does the presented risk driver actualize in the case project. Answer with one of following options: Yes / No / Cannot estimate / Not relevant (Why?)

Q: Do the method checklists seem complete?
Q: Is the presentation accurate enough or would something need further explanation?

Assessing the risk profile
Next, we are going to assess project risk profile using the provided risk profiling tables, which prioritize the risks based on their impact and category.
Q: Is the presented risk profile reasonable for the examined project?
Q: Is the presentation of the risk profile easy to use and compatible to the reporting and current risk management practices in your project?
Q: Are the method presented in language that is familiar to you or is there some terms that you would not use yourself?

Prioritizing the risks and reviewing presented risk resolution patterns

Based on the project risk profile, we are next going to use risk resolution pattern table to assess which risks should be mitigated first.
Q: Do you think that the risk levels of different type of requirements in suggested resolution patterns are accurate (the resolution pattern order)?
Q: Do you think that the decision made based on the suggested resolution pattern would be correct and work in your project?

Reviewing the suggested risk resolution techniques
First asking own insights from the interviewee and then going through the risk resolution techniques provided for each type of risk.
Q: What kind of resolution techniques you would apply to the risks based on your own expertise?
Q: Introducing the risk resolution techniques table: are these techniques familiar to you?
Q: How many of the presented techniques you have used?
Q: Taking into account the project phase and used developed methodology, do you think that applying the suggested techniques would be possible?

Common discussion about applying the method
Q: Do you find the method easy to learn and use?
Q: Does this method provide you such information that is complete, accurate and useful in project decision making and in your everyday work?

Theme 3: Completeness, accuracy and understandability of the results

Q: Do you think that risks that can be identified with the method are meaningful when thinking about overall project success?
Q: Have you identified corresponding risks earlier with your current risk management practices?
Q: Did the model bring up some new requirements risks that have not been identified with current risk management practices?
Q: Have you identified some other requirements risks in your project, which were not taken into account by the method?

Theme 4: Method feasibility to project use and usefulness of the results

Q: Do you think the method would be easily applicable to your current project?
Q: Could you plan needed requirements risk management actions by using the method?
Q: Would you estimate that using the method would improve the overall outputs or decision-making of requirements process?
Q: Would you estimate that using the method would have positive impact to your project team's productivity and/or overall project success?
Q: Do you think you are going to use the presented method in your project in future? If not, why?

Summary and ending

Q: Do you still have some thoughts or ideas you would like to share or discuss about?
Q: Would you have any additional feedback of the model or greetings to the research team?

APPENDIX B

Requirements resolution technique listing and categorization presented in the initial Continuous Requirements Risk Prioritization

Table 31 The initial requirements risk resolution techniques list and categorization adapted from Mathiassen et al. (2007: 594–596) by Tuunanen et al. (2016).

Name	Specification	Experimentation	Discovery	Prioritization
Affinity technique			*	
Aspect mining in requirements specification			*	
Attributed goal-oriented analysis	*		*	
Behavior analysis	*		*	
Box structure specification and design	*			
Brainstorming			*	
Business information analysis and integration technique	*		*	
Business process planning (BSP)	*		*	*
Card sorting			*	*
Cognitive mapping			*	
Contextual design	*		*	*
Cooperative prototyping		*		
CREV	*		*	
CREWS	*		*	

Critical success factors			*	*
Data flow diagram	*			
Decision analysis			*	
Delphi method			*	*
Deriving requirements from existing system			*	
Domain specific modeling	*			
EasyWinWin			*	*
Email/bulletin board			*	
Ends/means analysis			*	
Entity-relationship modeling	*			
Facilitated team			*	
Focus group			*	
Future analysis			*	
Goal modeling oriented requirements elicitation	*		*	
Goal oriented approach	*		*	
Group support systems and strategic business objectives			*	*
Guided brainstorming			*	
Human, social and organizational requirements elicitation			*	

Inquiry cycle model – structure and describe requirements discussions	*		*	
Joint application design		*	*	
KAOS	*			
Laddering			*	
Lyee	*			
Machine rule induction	*			
Marketing and sales			*	
MIS intermediary	*		*	
Multidimensional data models	*			
Multidimensional scaling	*			
Nominal group technique			*	*
Normative analysis			*	
Object oriented Z	*			
Open interview			*	
Open systems task analysis			*	
Participatory design		*	*	
Petri nets	*			
Petri nets combined with use cases	*		*	
Precision model			*	
Prime-CREWS	*		*	

Process analysis			*	
Protocol analysis			*	
Prototyping		*		
Quality function deployment	*			*
Repertoire grids			*	
Requirements generation model			*	
Requirements prototyping		*		
Requirements workshops			*	
Rich pictures	*		*	
Scenario-based requirements elicitation	*		*	
Semantic maps			*	
Socio-technical analysis			*	
State charts	*			
Strategic business objectives			*	*
Strategy set analysis			*	
Structured group elicitation method			*	
Structured interview			*	
Structured walkthroughs			*	
Support line			*	
Surveys			*	

Teach-back interview			*	
Testing		*	*	
Text analysis			*	
Trade show		*	*	
Usability lab		*		
Use cases			*	
Use of video in requirements elicitation			*	
User group			*	
User-interface prototyping		*	*	
Variance analysis			*	
VDM ++,VDM-SL	*			
Warnier-Orr diagrams	*			
Z	*			