Teemu Räsänen

REQUIREMENTS ENGINEERING FAILURE FACTORS IN SOFTWARE PROJECTS

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2021

TIIVISTELMÄ

Räsänen, Teemu
Vaatimusmäärittelyn epäonnistumistekijät sovellusprojekteissa
Jyväskylä: Jyväskylän yliopisto, 2021, 60 s.
Tietojärjestelmätiede, pro gradu -tutkielma
Ohjaaja: Siponen, Mikko

Vaatimusmäärittely on olennainen osa sovellusprojekteja ja sen onnistumisella on merkittävä vaikutus kaikkiin muihin sovellusprojektin osa-alueisiin. Tässä tutkielmassa pyritään tuottamaan lisää tietoa vaatimusmäärittelyyn liittyvistä epäonnistumistekijöistä kirjallisuuskatsauksen ja empiirisen tutkimuksen avulla. Kirjallisuuskatsauksessa käydään läpi olemassa olevia sovellusprojektien epäonnistumistekijöitä käsitteleviä tutkimuksia ja selvitetään, mitkä tutkimuksissa mainituista epäonnistumistekijöistä koskevat vaatimusmäärittelyä. Tutkielman empiirinen osa pohjautuu laadulliseen haastattelututkimukseen, jossa selvitetään, mitä vaatimusmäärittelyyn liittyviä epäonnistumistekijöitä IT-alan asiantuntijat pitävät merkittävimpinä. Tutkielman kirjallisuuskatsauksessa löydettiin aiemmista tutkimuksista yhteensä 17 vaatimusmäärittelyyn liittyvää epäonnistumistekijää, ja empiirisessä tutkimuksessa toteutetuista seitsemästä haastattelusta epäonnistumistekijöitä tunnistettiin kahdeksan. Näistä kahdeksasta epäonnistumistekijästä neljä mainitaan kirjallisuuskatsauksessa läpikäydyissä tutkimuksissa ja neljä on kokonaan uusia.

Asiasanat: vaatimusmäärittely, sovellusprojektit, epäonnistumistekijät

ABSTRACT

Räsänen, Teemu
Requirements Engineering Failure Factors in Software Projects
Jyväskylä: University of Jyväskylä, 2021, 60 pp.
Information Systems Science, Master's Thesis
Supervisor: Siponen, Mikko

Requirements engineering is an essential part of software projects, and its success has a significant impact to all other project activities. This study attempts to produce more information regarding requirements engineering failure factors by the means of a literature review and an empirical research. In the literature review, existing scientific papers addressing software project failure factors are examined to find out which of the mentioned failure factors concern requirements engineering. The empirical part of the study is based on a qualitative interview research, where it is investigated what requirements engineering failure factors are the most prominent according to IT professionals. A total of 17 requirements engineering failure factors were identified in the literature review, and eight failure factors were disclosed from the seven interviews of the empirical research. Four of these eight failure factors are mentioned in the scientific papers inspected in the literature review, whereas four are entirely novel.

Keywords: requirements engineering, software projects, failure factors

FIGURES

FIGURE 1 Proportion of failure factors concerning requirements engineering in past studies 8
FIGURE 2 Requirements engineering process 18
FIGURE 3 Dimensions of requirements engineering techniques 24
FIGURE 4 Content analysis process 34

TABLES

TABLE 1 IS project failure factor categories 14
TABLE 2 Common requirements elicitation techniques 23
TABLE 3 Requirements engineering failure factors in existing literature 27
TABLE 4 Requirements engineering failure factor categories 28
TABLE 5 Interviewees of the semi-structured interviews 35
TABLE 6 Interview 1 results 36
TABLE 7 Interview 2 results 37
TABLE 8 Interview 3 results 39
TABLE 9 Interview 4 results 40
TABLE 10 Interview 5 results 41
TABLE 11 Interview 6 results 42
TABLE 12 Interview 7 results 43
TABLE 13 Compiled failure factors and their occurrences in the interviews 44
TABLE 14 Failure factors identified in the interviews divided into categories . 44
TABLE 15 Requirements engineering failure factors identified in existing literature and in the empirical research 48

TABLE OF CONTENTS

TIIVISTELMÄ
ABSTRACT
FIGURES
TABLES

1 INTRODUCTION 7
1.1 Motivation for Research 7
1.2 Research Questions 9
1.3 Structure 9

2 SOFTWARE PROJECT FAILURE 11
2.1 Defining Failure 11
2.2 Incidence and Cost of Failure 12
2.3 General Failure Factors 13

3 REQUIREMENTS ENGINEERING 16
3.1 Definition 16
3.2 Requirements Engineering Process 17
3.3 Traditional Versus Agile Software Development 19
3.4 Requirements Elicitation Techniques 22
3.5 Frameworks and Methodologies 24
3.6 Success Factors 25
3.7 Failure Factors 27

4 RESEARCH DESIGN 30
4.1 Research Method 30
4.2 Data Analysis 32

5 EMPIRICAL RESULTS 35
5.1 Interview Results 35
5.1.1 Interview 1 36
5.1.2 Interview 2 37
5.1.3 Interview 3 38
5.1.4 Interview 4 40
5.1.5 Interview 5 40
5.1.6 Interview 6 42
5.1.7 Interview 7 42
5.2 Compiled Results 44

6 DISCUSSION 46

6.1 Theoretical and Practical Implications 49
6.2 Limitations 49
6.3 Future Research 50

7 CONCLUSION 51

REFERENCES 52

APPENDIX 1 INTERVIEW GUIDE (ENGLISH) 57

APPENDIX 2 INTERVIEW GUIDE (FINNISH) 59

1 INTRODUCTION

1.1 Motivation for Research

Failure of software projects is an evergreen topic of information systems research. A considerable portion of software projects ends up failing, which causes annual expenses measured in billions to corporations worldwide (Charette, 2005; Wood-Harper, 2007). Understanding the reasons behind software project failure can make avoiding failure easier, and many so-called failure factors have been identified in past studies. Software project failure factors have been researched quite extensively in general, but failure factors related to one sometimes overlooked part of software projects - requirements engineering - has not been a focal point of many research papers.

When looking at past research regarding failure factors in software projects, it is apparent that failure factors related to requirements engineering are a minority among all identified failure factors. When looking at six frequently cited failure factor studies by Chow and Cao (2008), Clancy (1995), Ebad (2020), Goedeke, Mueller and Pankratz (2017), McManus and Wood-Harper (2007), and Verner, Sampson and Cerpa (2008), requirements engineering related failure factors make up only 6,3 % of all failure factors identified in the studies (figure 1). Such small proportion of requirements engineering failure factors seems contradictory with the significance of requirements engineering in software projects: all other software project activities are dependent on it, and it can be even argued that it is the single most critical process in software development (Shams-Ul-Arif & Gahyyur, 2009; Pandey, Suman & Ramani, 2010). The low representation of requirements engineering in the studies could hint that requirements engineering doesn't receive much attention compared to other software project activities, or that there are requirements engineering related failure factors that have not been identified yet.

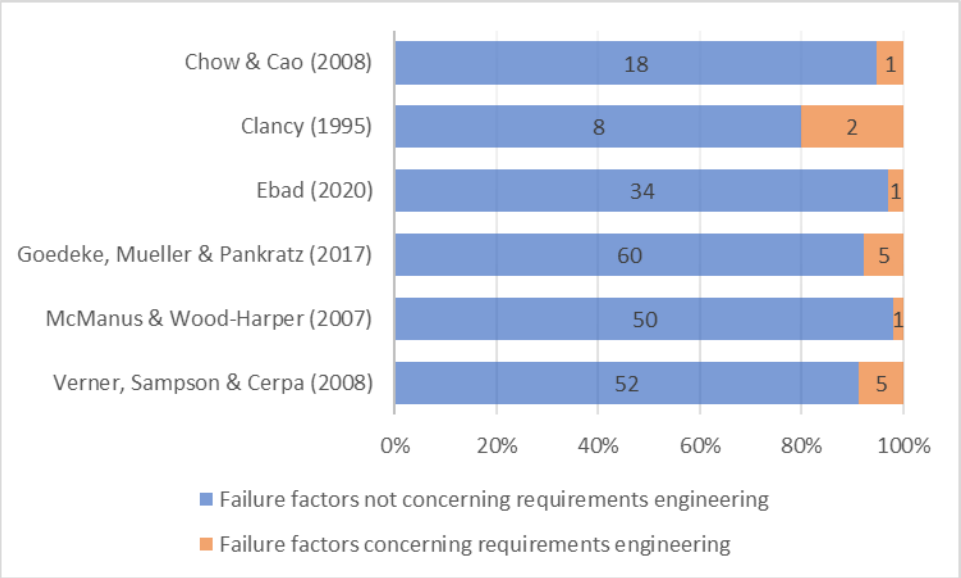

FIGURE 1 Proportion of failure factors concerning requirements engineering in past studies

The goal of requirements engineering is to define and describe the purpose of the software that is being developed. It is an important part of software projects because it paves the way for other project activities, such as designing and developing the actual software that is the end product of the project. Thus, failure in requirements engineering can potentially affect the whole project. Multiple failure factors exist that are specifically related to requirements engineering and increasing the information regarding them is a step towards better being able to avoid them.

The purpose of this study is to provide a comprehensive view of what requirements engineering failure factors are used in existing literature, as well as to give insight about what IT professionals think are the most prominent requirements engineering failure factors. The results of the study have both practical and theoretical implications. In practical terms, knowledge about requirements engineering factors that increase the chance of software project failure can help avoiding these factors in real life situations. This information is useful especially for managers of software projects, but for other project team members as well. The results of the study also expand the existing theoretical knowledge about requirements engineering failure factors by giving a comprehensive general view about what failure factors are used in existing literature. From theoretical point of view, the empirical results of the study also introduce several requirements engineering failure factors that are not mentioned in the examined literature, thus hinting towards the need for additional research regarding the subject.

1.2 Research Questions

As pointed out in the previous chapter, there seems to be a shortage of studies that examine software project failure factors mainly from the viewpoint of requirements engineering. The following research questions were composed to address this research gap:

- RQ1: What requirements engineering failure factors are mentioned in existing studies addressing software project failure factors?
- RQ2: What are the most recurrent requirements engineering failure factors in software projects according to IT professionals?

The first research question is answered by conducting a literature review on past studies regarding software project failure factors. The results of the literature review give insight about what requirements engineering failure factors are used in existing studies that focus on software project failure factors in general. Seven scientific papers were closely analysed and a total of 17 requirements engineering failure factors were identified in our literature review.

For answering the second research question, an empirical study was conducted to find out what IT professionals think are the most frequently occurring requirements engineering failure factors in software projects. Seven IT professionals with various titles and professional experience ranging from 3 to 26 years were interviewed in a semi-structured manner. The empirical results give an up-to-date view about the subject regarding the practitioners on the field. Eight requirements engineering failure factors were derived in our empirical study.

1.3 Structure

The structure of the thesis is as follows. After this introduction, there is a literature review that attempts to answer the first research question about what the most recurrent requirements engineering related failure factors in software projects are according to existing research. The first literature review chapter discusses project failure in general. This includes addressing the varying definitions of project failure found in existing literature, examining the incidence and cost of software project failure, and finally analysing past studies to compile a list of common failure factors in software projects. The following chapter presents requirements engineering, including how it manifests differently in traditional and agile software development, as well as what techniques, frameworks and methodologies are related to it. Also, failure factors and success factors that are specific to requirements engineering are identified, listed, and categorised. Subsequently, an empirical study is displayed, including its research method, methods for data collection and analysis, and results. After the research has been presented, there is a section reserved for discussion about the study, in-

cluding examination of its reliability and validity, limitations, and options for future research. Last, there is a conclusion chapter for the study.

2 SOFTWARE PROJECT FAILURE

2.1 Defining Failure

Before looking at factors leading to software project failure, it is essential to define what failure means in this context. Studies that address software project failure use varying criteria to define failure, and as pointed out by Hughes, Rana and Simintiras (2017), there seems to be no consensus about its exact definition. Ibraigheeth and Fadzli (2019) analysed several studies regarding software project failure and identified that nearly all software projects that were labelled as "failed" had at least one of the following attributes:

- Project was cancelled or its end product was never released.
- End product was lacking in quality.
- User requirements were not met.
- Budget was exceeded.
- Project was not completed on schedule.

It can be inferred that software project failure can happen in several areas regarding monetary and temporal limitations, user requirements and quality of the end product.

From their first Chaos Report onward, Standish Group (1994) has used a tripartite categorization system, where projects are classified as either failed, challenged or successful. In this system, a project belongs to the category "failed" only if it is cancelled completely. A challenged project is one that exceeded its budget or schedule, or whose final product needed to be stripped of features that were included in the original plan. A project is seen as successful only if it does not fall into either of these two categories and is delivered on budget, on time and within defined requirements.

It is worth mentioning that the failure criteria used in the Chaos Reports have been questioned on the basis that people might have a tendency to make too optimistic predictions regarding budgets and schedules in the beginning of projects (Eveleens & Verhoef, 2009). This raises a question if a project should be

considered failed if it is unable to meet these overwhelmingly optimistic budgets and schedules that are set to it.

Goedeke, Mueller and Pankratz (2017) defined eight dimensions of an IS project and stated that failure in any of them leads to the whole project being labelled as failed. These dimensions are time, cost, quality, quality of the project management process, stakeholder satisfaction related to process, project goal, project purpose and stakeholder satisfaction related to product. In other words, success in seven dimensions is nullified by failure in just one. It is worth noting that the researchers do not recommend using this strict fashion of labelling a project failed in general, and they justify using it in their study to be able to accurately track the impact that different factors have regarding project failure.

It is also possible to define failure based on perceptions of people involved in the project. In their study about factors leading to software project failure Verner, Sampson and Cerpa (2008) deemed a project failed if an interviewed developer that worked for the project considered it failed. This style of defining failure seems to be more subjective in nature than those mentioned earlier, and its results are dependent on who was asked about the project: two developers could have different opinions about whether a project failed or not. However, this methodology could be useful when interviewing a vast amount of people about projects and not having enough time to gather relevant data about the projects' budgets, schedules and other properties that are needed to be able to objectively assess whether the projects failed or not.

2.2 Incidence and Cost of Failure

A large portion of software projects end up failing, at least to some extent. According to the first Chaos Report of Standish Group (1994), the percentage of successful projects at the time of the research was as low as 16,2 %. Their corresponding report published in 2013 predicated that this number had risen to 29 % in 2004 and 39 % in 2013. Sauer, Gemino and Reich (2007) examined project success rates in the UK and gave more encouraging results: out of the five performance categories used in their study, the two most successful ones ("good performers" and "star performers") included 67 % of the surveyed projects. Comparing these results with those of Chaos Report is not straightforward because of different categories used in the studies. Also, as pointed out by Jørgensen and Moløkken (2005), the 1994 Chaos Report's findings are not in line with other studies published in the same period of time, and the results may give an unrealistically grim view of project failure rates.

Moneywise, failure of software projects causes annual expenses measured in billions to corporations worldwide (Charette, 2005). According to McManus and Wood-Harper (2007), the cost of software project failure within the European Union alone was 142 billion Euros in 2004. The total cost of failure is a sum of direct and indirect expenses; in addition to direct monetary costs, failure of a software project can negatively affect the company's image, business value,

marketability, and perceived satisfaction of customers (Hussain, Mkpojiogu & Mohmad Kamal, 2016). These effects can cause companies to suffer indirect long-term losses, which may not be easily quantified or measured.

Software project failure can also negatively affect the company's future software projects. In case a software project failed because of poor software quality, the created software might be difficult to maintain or expand. Also, if the created software is a part of a larger system complex and performs poorly, other system parts might suffer from the emerged bottlenecks. These phenomena can then cause additional costs long after the failed software project has been completed. (Zamudio, Aguilar, Tripp & Misra, 2017.)

2.3 General Failure Factors

Software projects can fail for a grand number of reasons which can be related to various project aspects ranging from technology and requirements to skillset of project members and communication during project (Goedeke et al, 2017). Failure factors are factors that increase the chance of project failure. They have been studied extensively in the past, especially from the viewpoint of IS projects.

Goedeke et al (2017) identified a total of 65 IS project failure factors among 23 failed IS projects they examined and grouped the failure factors into 13 categories (table 1). The six most frequently occurring failure factors were spread among several categories: planning (poor project planning), scope (significant integration & large-scale project), skills (lack of required skills & lacking project management skills) and technology (unrealistic technological expectations).

TABLE 1 IS project failure factor categories (Goedeke, Mueller & Pankratz, 2017)

Failure factor category	Description
Scope	Factors related to the overall objective of the project
Planning	Factors related to the estimating and planning of the project
Requirements	Factors related to the definition and analysis of the IS requirements
Technology	Factors related to the used technology and technological environment
Project process/controlling	Factors occurring during project execution and monitoring
Skills	Factors related to the skill level, knowledge, and capabilities of the project team
Contractor	Factors related to the relationship with contractor
Contract	Factors related to the project contract
Change management	Factors related to the management of the implementation and introduction of the IS in the customer organization
Communication	Factors related to communication or coordination within the project
Structure/Culture	Factors related to structure and culture of customer organization
Stakeholders	Factors related to problems and challenges with the stakeholders
Environment	Factors imposed by the environment of the project

Verner et al (2008) analysed 70 failed software projects and investigated the incidence of 57 failure factors in them. The amount of failure factors per observed project ranged from 5 to 47. The researchers conveyed that the most prevalent failure factors among the surveyed projects were:

1) Delivery date impacting the development process.
2) Underestimation of the project.
3) Inadequate re-assessment, control, and management of risks.
4) Not rewarding staff for working long hours.
5) Making delivery decision without sufficient information about requirements.
6) Staff having a displeasing experience working on the project.

McManus and Wood-Harper (2007) listed several key reasons why software projects are cancelled - which in this context equals to failure. The researchers examined a total of 214 software projects, of which 51 were cancelled (and thus failed). According to these numbers the failure rate of the examined projects was 23,8 %. The researchers divided the key failure reasons into three categories: business reasons, management reasons and technical reasons. 53 % of surveyed (cancelled) projects were cancelled mainly because of management reasons (e.g., insufficient risk management or loss of key personnel), 27,4 % because of technical reasons (e.g., inappropriate coding language or poor systems integration) and 19,6 % because of business reasons (e.g., poor requirements management or

misuse of financial resources). The researchers listed a total of 51 software project failure factors.

Chow and Cao (2008) examined existing agile software development related literature and gathered software project failure factors mentioned in them. The researchers were able to identify 19 failure factors, which they grouped into four categories: organizational, people, process and technical. The amount of failure factors listed by Chow and Cao (2008) was significantly lower than in other reviewed studies, which might be explained by the fact that the researchers focused solely on studies related to agile software development.

3 REQUIREMENTS ENGINEERING

3.1 Definition

All activities of a software project are founded on the project's requirements. Requirements are like a blueprint of the system that is being developed, and they direct where the project should be headed and what qualities the end product should have. Institute of Electrical and Electronics Engineers (IEEE) gives the following definition for a requirement:

1. A condition or capability needed by a user to solve a problem or achieve an objective.

2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.

3. A documented representation of a condition or capability as in (1) or (2).

(IEEE, 1990)

While the definition of IEEE is widely cited and considered a classic definition of a requirement, Glinz (2011) alters it slightly in order to make it more modern and compact:

1. A need perceived by a stakeholder.

2. A capability or property that a system shall have.

3. A documented representation of a need, capability or property.

(Glinz, 2011)

Requirements can be divided into functional and non-functional requirements. As mentioned by Van Lamsweerde (2009), functional requirements define what properties the developed software should have and how it should behave in its context. Non-functional requirements on the other hand describe softer prerequisites for the software, such as usability, security, performance, and maintainability (Chung & do Prado Leite, 2009; Pandey, Suman & Ramani, 2010).

Requirements engineering (RE) is an activity that is a part of software development. Nuseibeh and Easterbrook (2000) state that its function is to clarify and describe the purpose of the system that is being developed. This is achieved by recognizing relevant stakeholders and their needs, and by documenting these needs in a way that enables analysing them, communicating about them, and finally fulfilling them in the implementation phase of the project (Nuseibeh & Easterbrook, 2000). Shams-Ul-Arif and Gahyyur (2009) describe RE as "the initial step of software development activity in which the requirements from the customer are elicited and documented" and emphasize its significance for software project success, as all other project activities are dependent on it.

3.2 Requirements Engineering Process

RE process can be split into smaller series of action, but as mentioned by Zamudio et al (2017), multiple ways to do so are presented in the literature. One of the broader ways is to divide RE into requirements development and requirements management (Hussain et al, 2016). As Pandey et al (2010) elaborate, requirements development consists of discovering, analysing, validating, and documenting requirements, while requirements management is about traceability and change management of requirements. Requirements development and management are both iterative processes; requirements need to be gathered, refined, documented, and managed in multiple cycles to ensure having up-to-date and high-quality requirements throughout the project (Wiegers & Beatty, 2013).

Requirements development sees its biggest usage at the start of the project whereas requirements management becomes more essential as the project moves forward (Hussain et al, 2016). This is especially true in traditional software development, where requirements are often mainly composed early in the project's lifecycle (De Lucia & Qusef, 2010). However, requirements development and requirements management both span throughout the project. When the project reaches its implementation and integration phases, requirements still need attention as their changes need to be addressed and status updated according to the phase of the project (Hoffmann, Kuhn, Weber & Bittner, 2004).

The RE process can also be expanded further. Zamudio et al (2017) split RE activities into five parts. Their definition of the process differs from the aforementioned broad definition in a way that requirements development is split into five parts: requirements elicitation, requirements analysis, requirements specification and requirements validation. Requirements management

remains as a standalone part of the process. The RE process as it is defined by Zamudio et al (2017) can be seen in figure 2 below.

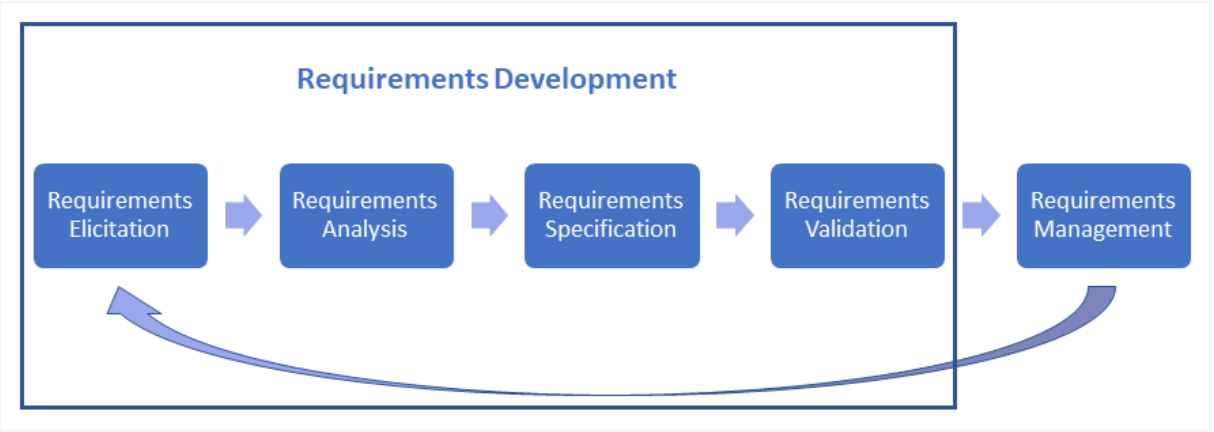

FIGURE 2 Requirements engineering process (Zamudio et al, 2017)

Requirements elicitation consists of discovering problems that need to be solved, identifying relevant stakeholders, and determining objectives of the software that is being built (Zamudio et al, 2017). Hickey and Davis (2004) state that the goal of requirements elicitation is to understand the needs of stakeholders. These needs are then translated into requirements. There are many different requirements elicitation techniques that can be used to extract viable requirements from the project's stakeholders. (Hickey & Davis, 2004.)

In requirements analysis phase, a deeper understanding of each requirement is achieved through several means. Functional requirements are derived from existing requirements information, and higher-level requirements are decomposed into smaller and more detailed requirements. Also, the implementation priorities are negotiated in this phase. (Wiegers & Beatty, 2013.) For example, models or prototypes can be created to aid in producing complete and satisfactory requirements. During requirements analysis the understanding of the organization's structure and goals is sharpened, and it becomes more clear what data is needed. (Zamudio et al, 2017.)

Requirements specification is a task where analysed requirements are documented to be used by the project team. Traditionally this has been most simply achieved by creating a single text document containing all requirements during early stages of the software project. (Firesmith, 2003.) There are also more intricate requirements specification techniques such as use case modelling, templates, and global analysis (Bass et al, 2006). According to Wiegers and Beatty (2013), the core principle of requirements specification is translating the gathered user needs into actual requirements that can be comprehended, used, and reviewed by their audiences.

Requirements validation is about verifying that the produced requirements accurately reflect the needs of stakeholders (Zamudio et al, 2017). Kamalrudin and Sidek (2015) state that the requirements validation process should consider the consistency, completeness, and correctness ("the 3 C's") of the requirements to catch errors, such as incompleteness and inconsistency. Require-

ments validation also helps determining that the project's end product is complete and that it fulfils the stakeholders' original requirements. (Kamalrudin & Sidek, 2015.)

Requirements management consists of recognizing changes to requirements and carrying out version control (Zamudio et al, 2017). It includes structuring and administration of information that is generated by other RE activities during the full lifecycle of the project (Hoffmann et al, 2004). Requirements management is most effective when it is conducted with the help of a requirements management tool, which collects the requirements into a repository. Such tools enable efficient collaboration among the project team and allow automation of requirements management processes, such as change history management and request approval. (Pandey et al, 2010.)

3.3 Traditional Versus Agile Software Development

In traditional software development, RE usually takes place at the beginning of a project: its purpose is to extensively describe what is wanted from the project before moving on to designing and developing the required software (De Lucia & Qusef, 2010). In this context RE appears to be an easily distinguishable part of a development project that has a beginning and an end. This traditional way of carrying out RE mainly at the beginning of a project has its downsides, since project requirements often evolve and do not remain static throughout the project. Reformation of requirements during a project can be driven by technological or industrial advancements, or simply by customers changing their minds about what they want from the project. Thus, it may not be sensible to try defining all-encompassing requirements before starting the project's design and implementation phases, as it may result in early defined requirements being partially or completely useless later in the project. (De Lucia & Qusef, 2010.)

Software development began shifting towards agile methods at the beginning of 2000's (Dingsøyr, Nerur & Balijepally, 2012), and consequentially the role of RE started to become increasingly less clear (Cao & Ramesh, 2008). Traditional and agile software development differ in how RE is conducted and at which part of a project it takes place. When in traditional software development RE usually happens at the beginning of a project (De Lucia & Qusef, 2010), Paetsch, Eberlein and Maurer (2003) state that agile development projects do not have a stiff chronological structure where RE would occur during a specific project phase. As Hussain et al (2016) put it, RE in agile software development is an "incremental and iterative process" that is "performed in parallel with other software development activities". Hence, requirements engineering spans throughout the project and emerges on every development cycle. As follows, requirements of developed software can change even near the very end of a development project (Paetsch et al, 2003).

Paetsch et al (2003) point out that some principles of traditional RE can be used in agile RE as well. As a notable similarity between the two worlds, the

elicitation, analysis, and validation phases exist in both of them. Also, both traditional and agile RE value the importance of customer involvement, even though it is more prevalent in agile RE. However, the two schools of thought have a couple of fundamental differences that cause contrast between the associated RE processes. First, agile development is rather adaptive than predictive, which can be an advantage when there are frequent changes during the project. Second, agile development focuses more on people than on processes. This means that agile methods prefer relying on the project team's expertise and direct collaboration rather than strict document-based processes. (Paetsch et al, 2003.)

While maintaining an extensive requirements documentation is a part of the traditional development process, in agile development it is replaced with periodic communication between the project team and customer (Ramesh et al, 2010). It is an assumption underlying agile development that developing encompassing documentation is counter-productive, partially because requirements can change later in the project, and the documentation might not be kept up to date (Turk, Robert & Rumpe, 2005).

As mentioned by Sillitti and Succi (2005), requirements are more strongly emphasized in agile software development than in traditional software development. This manifests during iterations of agile projects, where the most important requirements brought up by the customer are concentrated on. It is typical in agile development to implement high priority features early in the project to allow the customer to realize most business value (Ramesh et al, 2010).

In traditional RE most of the requirements are elicited before starting the development phase of the project, whereas in agile RE requirements are formed iteratively (De Lucia & Qusef, 2010). As mentioned by Sillitti and Succi (2005), it is often difficult to correctly predict all necessary requirements in the early phases of the project. This can lead to early requirements becoming obsolete later in the project whereas late-emerging requirements may turn out to be paramount. Requirements that later become useless may cause resources to be wasted in terms of spending time and money to develop unneeded features and afterwards maintaining these features. (Sillitti & Succi, 2005.)

Customer involvement is important in both traditional and agile RE, but it plays a bigger role in agile RE, as customer input is fundamental during the lifecycle of the whole agile project (Ramesh et al, 2010). Traditional RE is mostly based on the customer giving requirements for the wanted software at the beginning of the project, whereas in agile RE the customer is constantly giving feedback regarding the developed software. The customer's role in agile RE is so impactful that the customer can even be seen as a part of the project team. (Sillitti & Succi, 2005.)

Another difference between traditional and agile RE concerns who carries out requirements elicitation and management: in traditional RE only a part of the project team is primarily involved in carrying out these activities, whereas in agile RE it is the responsibility of the whole team to elicit and manage requirements (Sillitti & Succi, 2005). Cooperating with the customer requires cer-

tain proficiency from the project team; it is not enough to possess "hard skills" such as the ability to create excellent code, but the team members should be able to carry out constant dialogue with the customer and understand their needs (Cockburn & Highsmith, 2001). Agile RE can be said to be a mutual effort where the project team provides the customer with quality software, and the customer gives the project team feedback in return (Sillitti & Succi, 2005).

There are several practices that are characteristic to agile RE and thus differentiate it from traditional RE. As formulated by Schön, Thomaschewski and Escalona (2017), agile RE is more about using a list of prioritized requirements than using a requirements specification document. Cao and Ramesh (2008) examined 16 organizations that utilize agile approaches and identified seven agile RE practices within them:

1) Preferring face-to-face communication over written specifications
2) Iterative elicitation of requirements
3) Emphasizing requirement prioritization
4) Continual planning to cope with changing requirements
5) Prototyping
6) Test-driven development
7) Using review meetings and acceptance tests

Many of the identified practices quite accurately reflect the main principles of agile software development:

> 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, p. 1)

Preferring face-to-face communication over written specifications is a clear manifestation of valuing individuals and interactions over processes and tools, as well as is using review meetings and acceptance tests. Iterative elicitation of requirements and continual planning to cope with changing requirements reflect valuing responding to change over following a plan. Choosing not to emphasize written specifications corresponds to valuing working software over comprehensive documentation.

RE in agile software development poses its own challenges, many of which originate from the dynamic and fast-paced nature of the agile development process. According to Cao and Ramesh (2008), minimal documentation is one of these challenges. As documentation in agile projects if often limited to user stories and product backlogs, the project's traceability might be on a low level. Poor traceability in turn can lead to issues with scaling the project and introducing new members to the project team (Cao & Ramesh, 2008). Another

agile RE challenge mentioned by Cao and Ramesh (2008) is estimating the budget and schedule of a project. As further explained by Inayat, Salim, Marczak, Daneva and Shamshirband (2015), requirements are prone to change at any point of an agile software project, which can lead to initial estimates of the project's time and cost to become obsolete. Ramesh, Cao and Baskerville (2010) mention customer access and participation as a noteworthy agile RE challenge. This means that project's customers may not be available to negotiate requirements with, give feedback or answer questions regarding the project's requirements as often as it would be optimal in an agile environment. As continued by Inayat et al (2015), the consequence of low customer availability is an increase in rework needed in the project.

3.4 Requirements Elicitation Techniques

Multiple techniques exist that can be used in eliciting requirements of a software project. Some examples of requirements elicitation techniques are interviews, use cases, brainstorming and prototyping (Paetsch et al, 2003; Zamudio et al, 2017). As mentioned by Hickey and Davis (2004), the shared objective of all requirements elicitation techniques is to assist in understanding the needs of the project's stakeholders. Common requirements elicitation techniques and their strengths and weaknesses are compared in table 2.

TABLE 2 Common requirements elicitation techniques

Technique	Strengths	Weaknesses	Source(s)
Interviews	+ Enables getting a rich collection of information	- Interview data can be hard to analyse - Different interviewees may give conflicting information	Fricker et al. (2015), Paetsch et al. (2003), Zamudio et al. (2017)
Use cases	+ Easy to use	- Most effective when used early in the project	Fricker et al. (2015), Paetsch et al. (2003), Zamudio et al. (2017)
Communication analysis	+ Usage typically leads to complete models	- Complex to execute when compared to more straightforward techniques (e.g., use cases)	España et al. (2010)
Focus groups and workshops	+ Effective in extracting requirements + Can be helpful in solving conflicts among stakeholders	- Hard to carry out because multiple stakeholders need to be available at the same time	Fricker et al. (2015), Zamudio et al. (2017)
Brainstorming	+ Allows various viewpoints to be considered	- Is not typically useful in solving major issues	Zamudio et al. (2017)
Prototyping	+ Prototypes can sometimes be fully or partially used in the final product	- Does not function well as a solely used RE technique - Expensive in terms of time and cost	Fricker et al. (2015), Paetsch et al. (2003), Zamudio et al. (2017)

Jiang, Eberlein and Mousavi (2008) argue that context is an important factor when choosing which RE techniques to use in a project, as certain RE techniques work best in certain project environments. Despite this, people seem to have a tendency to choose what RE techniques they use based on their personal preferences rather than the context of the project at hand (Jiang et al, 2008). The same phenomenon is also mentioned by Tsumaki and Tamai (2005), who furthermore emphasize that choosing RE techniques that do not fit the ongoing project often leads to project failure. Hickey and Davis (2004) argue that analysts choose a specific requirements elicitation technique for a combination of four reasons:

1) The analyst does not know about other techniques
2) It is the favourite technique of the analyst
3) The analyst follows a methodology that instructs using the technique in the prevailing situation
4) The analyst innately understands that the technique is effective in the prevailing situation

Choosing an elicitation technique because of the fourth reason shows the most expertise by the analyst, which is valuable in terms of understanding the stake-

holders' needs. Most analysts do not possess enough professional know-how to make such qualified decisions, so they often end up relying on one of the first three reasons. (Hickey & Davis, 2004.)

3.5 Frameworks and Methodologies

Considering that using correct RE techniques is beneficial for a software project's success, it might be worthwhile to invest some time in choosing the said techniques. Frameworks and methodologies can be helpful in this matter. Basically, frameworks and methodologies take certain input - in this regard the project's context - and then give an output of which RE techniques should be used in the project. An example of such methodologies is Methodology for Requirements Engineering Techniques Selection (MRETS) by Jiang et al (2008), where an RE technique is recommended for a given project based on several attributes of the project. Some of these attributes are project complexity, time constraints, team size and product type. There are also other methodologies that similarly take certain project attributes and suggest an RE technique based on them.

Another example is Tsumaki's and Tamai's (2005) framework, where RE techniques are classified on two dimensions, after which their compatibility with a specific project is estimated based on the project's characteristics. The dimensions of RE techniques' evaluation are static-dynamic and closed-open (figure 3).

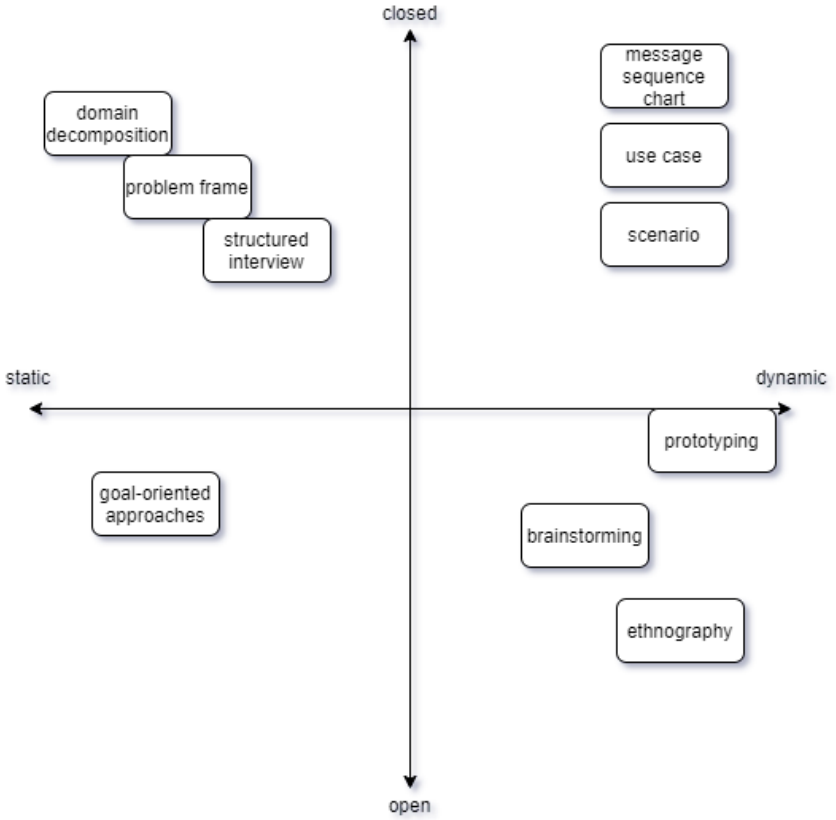

FIGURE 3 Dimensions of requirements engineering techniques (Tsumaki & Tamai, 2005)

A technique is considered static when it is used to extract requirements of a domain's invariant structure and dynamic when its focal point is on gathering requirements from a domain's mutable context. Closed-open dimension is used to define the technique's object space: a closed object space is one that is relatively known and stable, whereas an open object space is unknown and unstable. Five characteristics of a project are considered when analysing the compatibility of an RE technique with the said project: application domain, requirements engineer types, user involvement, information resources and requirements properties. The framework includes rules about how each of the characteristics affects the compatibility of a certain RE technique and the project. (Tsumaki & Tamai, 2005.)

3.6 Success Factors

As mentioned by Sudhakar (2012), identifying the critical success factors (CSFs) of a software project is a step towards making the project successful. Project manager's role is important in ensuring that CSFs are leveraged, but every member of the project team affects the success of the project. CSFs can be found regarding almost all activities of software projects, including RE. (Sudhakar, 2012.)

Defining success of RE and software projects in general is virtually as complicated as defining their failure, and as mentioned by Sudhakar (2012), various success definitions have been used in past studies. Logically, it can be presumed that a finished software project is successful if it did not fail. As formulated by Yousef, Gamal, Warda and Mahmoud (2006), a project's success is often defined by achieving project goals and features within established schedule and budget. This definition is – quite logically - close to the opposite of the earlier mentioned failure definition of Ibraigheeth and Fadzli (2019). Sudhakar (2012) states that a project's success is a combination of project management success and project product success. In other words, success can be seen to consist of quality of the end product that was created and also how well the creation process was carried out.

Since the role of RE is significant in terms of software projects' failure, it is also important regarding their success. As RE affects multiple aspects of a software project, its successfulness cannot be evaluated solely based on the logical end product of RE, which is the requirements specification. Fricker et al (2015) surveyed participants of multiple software projects (n=419) and recognized factors that were commonly used to measure RE's success in the projects. The study's findings give hint of what factors are most frequently used as indicators of RE's success in software projects. The said factors simultaneously act as RE's goals. The most prevalent RE goal identified in the study is establishing a shared understanding among project team and project's stakeholders. Good specification quality is an almost equally recurrent goal. Other common RE goals that were mentioned by at least 33 % of respondents are paving a clear

scope for upcoming development activities, performing RE efficiently, making users satisfied and delivering RE results on schedule.

Nasir and Sahibuddin (2011) identified a total of 26 critical software project success factors in 43 articles published between 1990 and 2010. Two of these success factors are related to RE:

1) Clear requirements and specifications
2) Frozen requirements

"Clear requirements and specifications" had a higher frequency among researched articles than any other success factor identified in the study; it was present in 60,5 % of the articles. The appearance rate of "frozen requirements" among the articles was 39,5 %. Requirements being frozen refers to them not changing throughout the project. This may not always be desirable, as there can arise situations where it is necessary to change existing requirements or introduce new ones. However, according to the researchers, having frozen requirements contributes to a project being successful, or in other words, it staying on budget and on schedule. (Nasir & Sahibuddin, 2011.)

Sudhakar (2012) carried out a literature review on past studies focusing on critical success factors of software projects and identified 80 success factors in them. Two of these success factors concern RE:

1) Realization of user requirements
2) Well-defined project requirements

The first one of these success factors is categorized as a product factor and the second one as a project management factor. The two requirements could be summarised as project's requirements being clear and the project fulfilling these requirements. (Sudhakar, 2012.)

Yousef et al (2006) list nine characteristics that are often present in highly successful software projects. Three of these characteristics concern RE:

1) Project requirements result in well-defined software deliverables.
2) There exists a single repository for requirements.
3) Requirements are accurate and complete before the project ends.

The first two of these characteristics were identified as the two most highly recurrent factors predicting the success of a software project, which underlines the importance of RE in regards of software project success. The third characteristic was found to be less significant, but still present in a high amount of successful software projects. (Yousef et al, 2006.)

Chow and Cao (2008) examined several case studies and meta-analyses about success factors in agile software projects and compiled a list of 36 success factors based on them. Two of the mentioned success factors are related to RE:

1) Project follows a requirement management process that is agile-oriented.
2) Project has a variable scope with emergent requirements.

A relatively small part of all success factors identified by the researchers concerns RE. The researchers focused solely on agile projects in their study, which might at least partially explain RE's low representation, since RE's role is not as clear in agile software development as it is in traditional software development

(Cao & Ramesh, 2008). This might make it harder to notice success factors that are directly induced by RE.

3.7 Failure Factors

As mentioned by Shams-Ul-Arif and Gahyyur (2009), all activities of a software project are dependent on RE. In the light of this information, it would make sense that several RE related software project failure factors could be found from existing studies. As it turns out, not much of the literature addressing software project failure factors seems to be doing so from the viewpoint of requirements engineering. Nevertheless, multiple RE related failure factors can be found from studies that cover software project failure factors in general.

For this study, a total of seven research papers regarding software project failure factors were analysed. The main criteria when choosing papers to include in the study was that the papers should address failure factors in general and not from the viewpoint of a specific dimension of software projects. The examined papers include 290 failure factors in total, of which 17 are related to RE. These failure factors are listed in table 3.

TABLE 3 Requirements engineering failure factors in existing literature

Failure factor	**Source study**
Ill-defined project requirements	Chow & Cao (2008)
Changing requirements	Clancy (1995)
Incomplete requirements	Clancy (1995)
Missing requirements	Ebad (2020)
Conflicting system requirements	Goedeke, Mueller & Pankratz (2017)
Incomplete requirements analysis	Goedeke, Mueller & Pankratz (2017)
Misinterpretation of requirements	Goedeke, Mueller & Pankratz (2017)
Unclear system requirements	Goedeke, Mueller & Pankratz (2017)
Unprioritized requirements	Goedeke, Mueller & Pankratz (2017)
Poor requirements management	McManus & Wood-Harper (2007)
Lack of frozen requirements	Schmidt, Lyytinen, Keil & Cule (2001)
Misunderstanding the requirements	Schmidt, Lyytinen, Keil & Cule (2001)
Customers/users did not make enough time for requirements gathering	Verner, Sampson & Cerpa (2008)
No method for requirements gathering	Verner, Sampson & Cerpa (2008)
No single requirements repository for requirements	Verner, Sampson & Cerpa (2008)
Project did not have good requirements at any stage	Verner, Sampson & Cerpa (2008)
Project not replanned after requirements changes	Verner, Sampson & Cerpa (2008)

There seems to be variance among studies regarding what kind of RE related failure factors are mentioned in them. Most failure factors brought up by Verner et al (2008) that can be associated with RE are about the requirements gath-

ering process being somehow faulty, with the only exception of "project did not have good requirements at any stage" being related to inferior quality of requirements. On the contrary, Goedeke et al (2017) mention only one failure factor that is related to the requirements gathering process (incomplete requirements analysis), whereas the rest of the brought-up failure factors are about inferior quality of requirements and incorrect usage of requirements.

Three of the examined studies include only one RE related failure factor. McManus and Wood-Harper (2007) mention a total of 51 failure factors, but only one of them can be associated with RE: "poor requirements management". Chow and Cao (2008) also name just one RE related failure factor – "ill-defined project requirements" - out of 19 failure factors identified in their study. The same applies to Ebad (2020); "missing requirements" was the only failure factor mentioned in the study that concerns RE.

The discovered failure factors can be divided into three categories based on how they are related to RE:

- Requirements development
- Requirements management
- Quality of requirements

It would be straightforward to categorize failure factors according to which part of the RE process they concern – requirements development or requirements management. However, a major portion of the identified failure factors describes the requirements themselves, not the RE process. Thus, it is necessary to include a third category: quality of requirements. Dividing the failure factors into categories makes it easier to comprehend what aspects of RE different failure factors concern and how the failure factors are distributed among different aspects. Failure factors and their corresponding categories can be seen in table 4.

TABLE 4 Requirements engineering failure factor categories

Category	**Failure factor**
Requirements development	No method for requirements gathering
	Customers/users did not make enough time for requirements gathering
	Incomplete requirements analysis
	Missing requirements
Requirements management	No single requirements repository for requirements
	Poor requirements management
	Project not replanned after requirements changes
	Lack of frozen requirements
Quality of requirements	Project did not have good requirements at any stage
	Unclear system requirements
	Misinterpretation of requirements
	Conflicting system requirements
	Unprioritized requirements
	Ill-defined project requirements
	Misunderstanding the requirements
	Changing requirements
	Incomplete requirements

The most of the listed failure factors are related to quality of requirements (53 %), whereas failure factors concerning requirements development (23,5 %) and requirements management (23,5 %) are in the minority. According to these results, it seems that the examined studies mostly focus on failure factors regarding the properties of compiled requirements, rather than failure factors concerning the RE process where the requirements are composed.

4 RESEARCH DESIGN

Poorly conducted RE is a common pitfall of software projects, but it has not been a focal point of many prior studies that have examined software project failure. The aim of this empirical study was to address this research gap by answering the following research question:

- RQ2: What are the most recurrent requirements engineering failure factors in software projects according to IT professionals?

As the area of interest has not been researched extensively before, an explorative approach was used in the study. Explorative research is typically applicable when the subject area does not have much existing information and the researcher wants to understand the problem better (Wohlin & Aurum, 2015).

In this chapter the study's research method is introduced, along with consideration about prerequisites for the interviews, such as what is an adequate sample size for the study. Next, the used data analysis method is presented. It is explained why the specific method was selected, and the whole data analysis process is explained in detail.

4.1 Research Method

The empirical research was conducted by using a qualitative methodology, and more precisely by carrying out semi-structured interviews. The interviewees were IT professionals with professional experience of software projects and requirements engineering. A qualitative methodology was selected because of the explorative nature of the research. As mentioned by Mason (2010), qualitative research is concerned of meaning rather than testing a hypothesis, which suits the scope of this study well.

Semi-structured interviews were chosen as the research method for a couple of reasons. First, they allow interview participants to explore topics that seem important to them (Longhurst, 2003). This kind of flexibility was considered important in this research, since it was possible that novel failure factors

that have not been mentioned in prior studies would emerge during the interviews. According to Bogdan and Biklen (1997), using open-ended questions allows the interviewees to answer the questions on their own terms, without the potential bias created by a fixed list of possible answers. Utilizing semi-structured interviews also makes it possible to gain a deeper understanding of surfacing failure factors, and as mentioned by Cohen and Crabtree (2006), semi-structured interviews give an opportunity to understand the researched topic in new ways.

There were several goals in selecting the interviewees for the research:

1) The interviewees should have worked with software projects.
2) The interviewees should have professional experience regarding requirements engineering.
3) The interviewees should represent multiple organizations.
4) The interviewees should have varying roles in software projects.

Goals 1 and 2 were set so that the chosen interviewees would have experience relevant to the research and goals 3 and 4 so that data could be gathered from multiple viewpoints regarding requirements engineering. The interviewer assessed the fulfilment of goals 1 and 2 individually for each interviewee, and goals 3 and 4 as the research progressed by comparing the backgrounds of new interviewees to those of prior ones.

An interview guide was prepared to assist in conducting the semi-structured interviews. As mentioned by Longhurst (2003), an interview guide can consist of either themes or complete questions. Questions were used in the interview guide of this research. English and Finnish versions of the interview guide were included because there were interviewees that preferred both languages.

Leech (2002) states that interviews should be started with questions that the respondents are likely to feel comfortable answering, whereas more complex questions should be handled later. This prerequisite was fulfilled in this research by first asking questions about the interviewees' professional backgrounds, as most interviewees can probably talk about their personal histories effortlessly. Questions requiring more thought were asked afterwards. Because requirements engineering as a concept might be tough to grasp, the questions were designed to gradually lead the interviewees into the subject. As mentioned by Longhurst (2003), it is not necessary to ask the interview questions in the planned order, as keeping the interviews conversational allows interviewees to elaborate on topics that they feel are important. However, at the end of the interview it should be checked that all questions have been dealt with to at least some extent (Longhurst, 2003).

The results of several scientific papers were examined when considering the adequate sample size of the study. Reaching saturation describes the situation where additional interviews do not provide new relevant information about the researched topic. As the purpose of this study was to discover new RE related failure factors – rather than test an existing hypothesis – reaching full saturation was not considered practical. According to Francis et al (2010), a sat-

uration level between 86 % and 92 % should be expected after six interviews. This saturation level seems reasonable for explorative research, so the goal was to gather at least six – preferably seven - interviewees for the study.

Galvin (2015) presented a formula to determine how many interviews need to be conducted to reach a certain probability that a belief or an attitude held by a certain proportion of the population emerges during the interviews. This formula is the following:

$$R = 1 - \sqrt[n]{1 - P}$$

R represents the portion of the population holding a certain belief or attitude, whereas *n* is the number of interviews conducted and *P* is the probability that a belief or an attitude emerges during *n* number of interviews. By conducting 7 interviews and pursuing a 95 % confidentiality level, we can calculate *R* as follows:

$$R = 1 - \sqrt[7]{1 - 0.95} = 0.3481636551311608\ldots \approx 0.35$$

The value of *R* being 0.35 means that by conducting 7 interviews there is a probability of 95 % that beliefs or attitudes held by 35 % of the population will emerge in the interviews. Because of the explorative nature of the research, we are satisfied with this number.

4.2 Data Analysis

The data analysis method used in this study is content analysis. Krippendorff (1980) defines content analysis as "a research technique for making replicable and valid inferences from data to their context". According to Elo and Kyngäs (2008), content analysis makes it possible to comprise categories out of observed words. Cavanagh (1997) further explains that the purpose of creating categories is to combine words and phrases with similar meanings together.

The main purpose of analysing the interviews is to find out what RE failure factors the interviewed IT professionals consider occurring most frequently in software projects. Thus, it is relevant to be able to identify failure factors based on the expressions of the interviewees. To achieve this, the expressions are compared with the failure factors that were found in our literature review. If an expression is seen to represent an existing failure factor, the corresponding failure factor is used. However, if an interviewee's expression describes a frequently occurring RE failure factor which was not found in our literature review, a new failure factor is composed.

Elo and Kyngäs (2008) elaborate that content analysis can be used in either deductive or inductive way. Deductive content analysis is feasible when testing an existing theory and when the used analysis is structured based on prior knowledge. Inductive content analysis is recommended if prior knowledge about the studied phenomenon is scarce, or if this knowledge is fragmented. (Elo & Kyngäs, 2008). As no existing theory is tested in this study, and since

past research papers do not provide comprehensive knowledge about RE failure factors, inductive content analysis is used.

Elo and Kyngäs (2008) present a process for deductive and inductive approaches of content analysis, which consists of three phases: preparation, organizing and reporting. The complete process can be seen in figure 4. Deductive and inductive approaches both share a similar preparation phase, but the organizing and reporting phases are different.

The preparation phase's first step is selecting the unit of analysis, which can be a word or a theme. As pointed out by Graneheim and Lundman (2004), whole interviews are a feasible unit of analysis, as long as they are not too small to be considered as a whole or too large to be kept in mind as a meaning unit's context. Elo and Kyngäs (2008) state that before starting the analysis, it should be decided whether to analyse latent content in addition to manifest content. According to Graneheim and Lundman (2004), latent content refers to the underlying meaning and nuances of the analysed content (tone of voice, laughter, etc.), whereas manifest content is its visible and most obvious part (text or spoken words).

Next in the analysis process of Elo and Kyngäs (2008), the researcher gets acquainted with the data at hand. This is done by reading the material multiple times while trying to truly understand its meaning. After the researcher has comprehended the data, analysis is carried out by using either a deductive or inductive approach. (Elo & Kyngäs, 2008.)

When using an inductive approach, the next step is to organize the data. This begins with open coding, where the analysed text is read and simultaneously marked with headings and notes. The purpose of open coding is to describe all sides of the content by writing down as many headings as necessary. Subsequently, the headings are gathered into coding sheets and categories are now freely developed. (Elo & Kyngäs, 2008.)

After open coding is finished, the resulting categories are arranged into higher level groupings to scale down the number of categories. This is done by taking similar categories and assembling them under the same grouping. (Elo & Kyngäs, 2008.) Dey (1993) emphasizes that categories cannot be put together just because they are similar, but rather they should share relevant properties that separate them from other categories.

The last step of the organising phase is called abstraction. The goal of abstraction is to create a broad description of the topic that is being researched by setting up categories. Similar categories are then arranged together into subcategories, and new categories are created to be their main categories. This is continued for as long as it is feasible. (Elo & Kyngäs, 2008.)

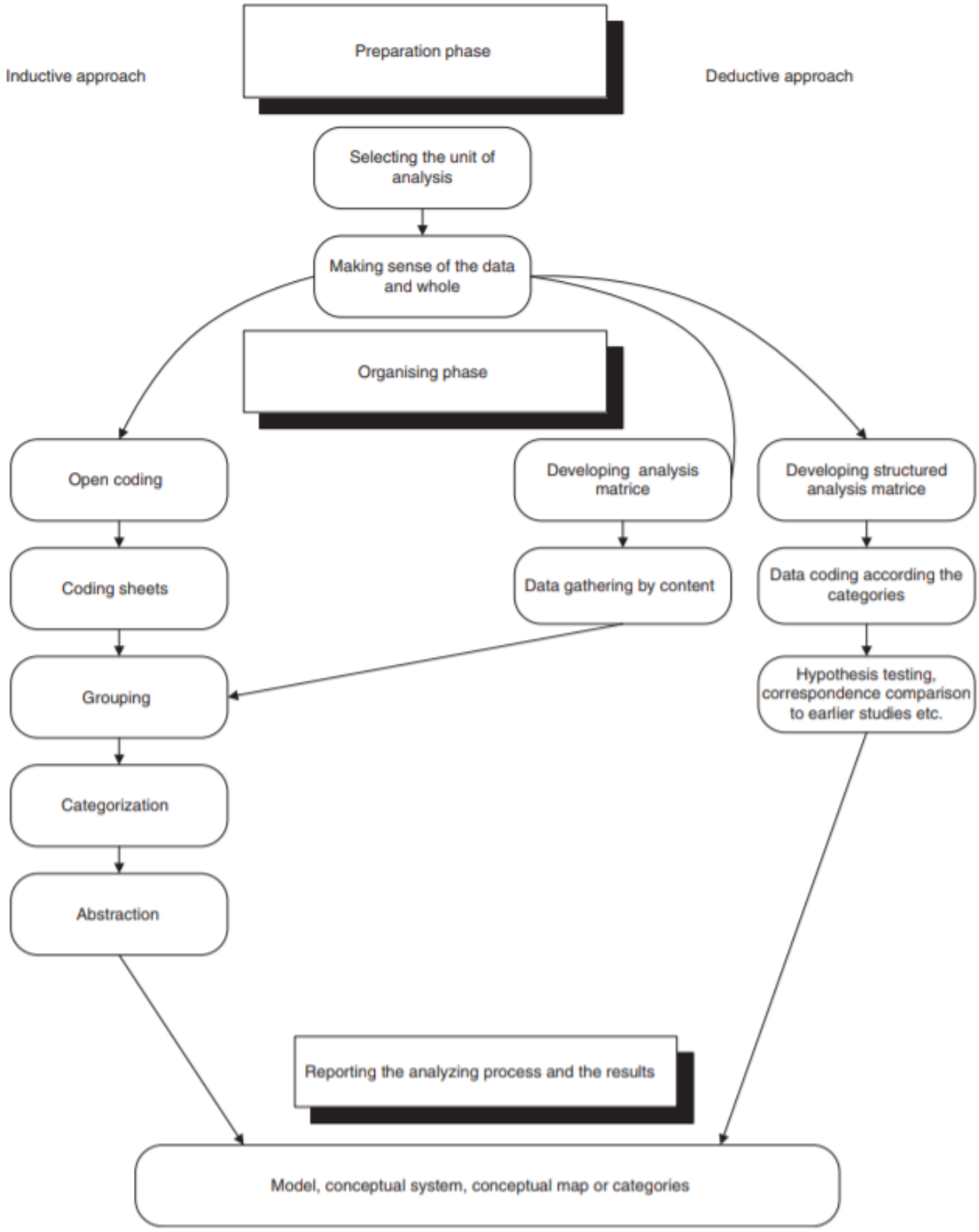

FIGURE 4 Content analysis process (Elo & Kyngäs, 2008)

5 EMPIRICAL RESULTS

The empirical results of the semi-structured interviews are presented in this chapter. First, the interviewees are introduced with information about their job titles and the amount of professional experience they have. Next, each interview is examined separately. The failure factors identified in each interview are shown in tables, where it can also be seen what the original expressions behind the derived failure factors were. Last, the compiled results of the interviews are presented, including a compilation chart of all failure factors that were identified in the interviews.

5.1 Interview Results

A total of seven people were interviewed for the study. The interviewees represent four IT companies and have varying viewpoints towards RE, as some are working with IT projects on a management level, and some take part in development themselves. The amount of professional experience among the interviewees ranges from 3 years to 26 years, with the average amount of professional experience being 20 years. A compilation of the interviewees can be seen in table 5.

TABLE 5 Interviewees of the semi-structured interviews

#	Job title	Professional experience (y)
1	Solution architect	14
2	Software developer	20
3	Manager	23
4	Platform architect	26
5	Software developer	3
6	IT consultant	26
7	Partnerships director	25

5.1.1 Interview 1

Interviewee #1 has worked with information technology for 14 years, of which he has been associated with software projects for 12 years. His professional experience consists mainly of working with a large telephony system in a specialist role. His current job title is solution architect. Four software project failure factors that are related to RE arose in the interview. These failure factors are listed in table 6 below.

TABLE 6 Interview 1 results

Original expression	Category	Failure factor
I was a part of a project where we just started working with virtually no requirements, and the project did not end well.	Requirements development	Missing requirements
Broad requirements were defined for the project, but there were no requirements for the sub tasks of the project.	Quality of requirements	Requirements not specific enough
Nobody had clear responsibility of handling requirements -- and the requirements were defined within a hassle of an e-mail conversation.	Requirements management	Poor requirements management
...we were at the point where we could start carrying out the migration, but then new requirements arose because of a recent business acquisition, which brought the project to a complete stop.	Requirements management	Changing requirements

The first often occurring failure factor mentioned by the interviewee was *missing requirements*. To give an example, the interviewee described a software project where the project team started working with very little requirements, which caused difficulties in focusing on correct tasks and taking the project towards the right direction. Because of missing requirements, the project team's resources were not used as efficiently as they could have been used with proper requirements.

The second common failure factor that emerged in the interview was *requirements not being specific enough*. According to the interviewee, requirements being too broad causes similar difficulties in software projects as having little to no requirements to work with, namely making it hard to know what exactly needs to be done in the project.

As an example of the third recurring failure factor, the interviewee brought up a project where nobody had a clear responsibility of requirements management, which caused the requirements to partially drown under other project related communication as nobody was keeping track of them. Also, the project's requirements were mainly defined in an e-mail conversation with multiple participants. This made it even harder to get a grasp of the project's requirements, as they were scattered around a lengthy e-mail thread. The failure factor that this description depicts is *poor requirements management*.

The last frequently occurring failure factor that came out in the interview was *changing requirements*. As an example of this failure factor, the interviewee described a software project which had been ongoing for several weeks and where everything had been planned according to existing requirements. Then suddenly new major requirements emerged, and the project had to be replanned. This brought the development phase of the project into a halt and caused a noticeable delay in the project's schedule.

5.1.2 Interview 2

Interviewee #2 works as a software developer for a medium-sized enterprise. He has roughly 20 years of professional experience, which consists of working for several IT companies. The interviewee has been associated with software projects for his whole career. Five RE failure factors were derived from the interview (table 7).

TABLE 7 Interview 2 results

Original expression	**Category**	**Failure factor**
Requirements are conducted too late. Things could work out better if more time would be used to developing requirements at the beginning of the project.	Requirements development	Requirements conducted too late
Too little or no time is used for requirements development…	Requirements development	Rushed requirements development
The requirements were half-baked…	Quality of requirements	Requirements not specific enough
…and they (requirements) kept changing throughout the project.	Requirements management	Changing requirements
Not enough people with different types of knowledge took part in the planning of the project.	Requirements development	Wrong people carrying out requirements development

The interviewee mentioned that it has been a common phenomenon in multiple software projects he has participated in that *requirements are conducted too late*. In such cases the software project has progressed too far with loose requirements when some new essential requirements emerge. The effect of conducting requirements too late resembles requirements changing during the project; replanning is often needed before continuing the development work, leading to the project being delayed.

Another recurring RE related mistake brought up in the interview was *rushed requirements development*. According to the interviewee, he has been a part of several software projects where very little time was used to develop requirements. The main problem caused by this has been requirements having poor quality.

Requirements not being specific enough was mentioned by the interviewee as he described a software project where the customer was not able to provide accurate and good quality requirements at any point of the project. The customer was not entirely sure what they wanted from the end product, which lead to the project's requirements being loose. The customer also changed their vision about the end product often during the project, which meant that the project had *changing requirements*.

The interviewee mentioned that software projects often do not have enough people with different types of professional expertise carrying out RE. This can lead to some important aspects of the project being neglected. For example, if only programmers are involved in RE, there might be a deficiency of requirements related to the UI or usability of the software. The failure factor that can be derived here is *wrong people carrying out requirements development*.

5.1.3 Interview 3

Interviewee #3 works as a manager for a large IT corporation. He bears responsibility and ownership over multiple information systems and has 23 years of professional IT experience. RE failure factors that emerged during the interview are listed in table 8.

TABLE 8 Interview 3 results

Original expression	Category	Failure factor
They (=requirements) are elicited poorly, or they are not elicited at all.	Requirements development	Missing requirements
People who are specifying requirements are unable to foresee the end result (of the project) closely enough, and do not possess enough technical knowledge about what needs to be done.	Requirements development	Wrong people carrying out requirements development
The "PowerPoint department" usually has quite broad requirements - - and usually it is only a scratch on the surface.	Quality of requirements	Requirements not specific enough
Along the way they (=requirements) change significantly for some reason.	Requirements management	Changing requirements

When asked about what are the most often recurring RE related mistakes in software projects, the interviewee first mentioned that requirements are often elicited poorly or "not elicited at all". Thus, the derived failure factor is *missing requirements*. This means that the project was carried out with some important requirements missing completely.

The second recurring failure factor that arose during the interview was *wrong people carrying out requirements development*. The interviewee mentioned that people who are specifying project requirements regularly lack technical knowledge that is necessary to understand the constraints of the project and to apprehend what is actually wanted from the end product. The interviewee named a specific group of people as the "PowerPoint department", which typically consists of middle managers that mainly understand the broad requirements but not the technical aspects of the project. According to the interviewee, when these middle managers conduct requirements elicitation, the resulting *requirements are not specific enough*.

Another frequent RE related failure factor mentioned by the interviewee was *changing requirements*. The interviewee stated that during many projects the requirements are not static and evolve a lot as the project progresses. This often causes increased workload to the project team through new feature requests and the need to make changes to existing software. Changing requirements can also hamper the original estimations regarding the budget and schedule of the project.

5.1.4 Interview 4

Interviewee #4 is a platform architect for a medium IT corporation. He has 26 years of professional experience, and his responsibilities have ranged from programming and database maintenance to taking care of software projects on a larger scale. Two RE failure factors could be extracted from the interview (table 9).

TABLE 9 Interview 4 results

Original expression	Category	Failure factor
Requirements are often defined in an undetailed manner, whereupon it is unclear for both parties what needs to be done.	Quality of requirements	Requirements not specific enough
People with enough experience and know-how should be present so that requirements of good quality can be conducted.	Requirements development	Wrong people carrying out requirements development

From the interviewee's point of view, it is a common phenomenon in software projects that *requirements are not specific enough*. According to the interviewee, requirements are often broad, which makes it difficult for both the vendor and the client to understand what exactly needs to be achieved by the developed software. This can cause struggles on the vendor's side to work efficiently towards the target. Moreover, the vendor spending excessive time on developing the software can lead to unfulfilled expectations and disappointment on the client's side.

The interviewee mentioned that occasionally there is too little suitable expertise among people conducting RE, or in other words, *wrong people are carrying out requirements development*. For example, a salesperson might not be the correct individual to conduct any extensive technical requirements for a project. The interviewee pointed out that people with enough relevant technical experience should take part in RE from very early stages of the project to increase the chance of ending up with quality requirements.

5.1.5 Interview 5

Interviewee #5 works as a software developer for a medium IT corporation. He has 3 years of professional experience, consisting mainly of developing software. Four RE failure factors were inferred from the expressions of the interviewee (table 10).

TABLE 10 Interview 5 results

Original expression	Category	Failure factor
Requirements might be broad, and they do not actually tell anything.	Quality of requirements	Requirements not specific enough
Fluctuating requirements - - cause more work for the project.	Requirements management	Changing requirements
The person who carries out requirements development is not a part of the project team and does not understand what is actually required (from the project).	Requirements development	Wrong people carrying out requirements development
The requirements are unclear for those who carry out the project - - and the customer does not know what they want.	Quality of requirements	Unclear requirements

The interviewee described that it is typical for software projects that *requirements are not specific enough*. According to the interviewee, this is especially true in large (hundreds of thousands of Euros) projects. The requirements being broad means that developers need to spend time in "decoding" them and splitting them into smaller, more comprehensive parts.

Changing requirements was mentioned by the interviewee as another recurring pitfall in software projects. When requirements fluctuate, the total amount of work needed for the project grows. The interviewee pointed out that this can be beneficial for the software vendor in case they are working with an hourly wage, as they can charge more from the customer. However, if the project is done with a piecework pay, the vendor generally does not want its workload to increase.

The interviewee stated that often the person who performs RE is not a part of the team that carries out the actual software development. Thus, the person might not possess enough technical knowledge to be able to produce adequate requirements. The interviewee mentioned that one area where this can cause problems is scheduling and budgeting the project. The identifiable failure factor here is *wrong people carrying out requirements development*.

Last common failure factor brought up by the interviewee was *unclear requirements*. Often requirements being unclear originates from the customer's inability to recognize and communicate explicitly what they want from the project's end product. This typically leads to ambiguous requirements, which are hard to interpret and cause additional work for the software developer.

5.1.6 Interview 6

Interviewee #6 works as a consultant for a large IT corporation. He is often involved in software projects and is specialized in requirements engineering. Two RE failure factors were remarked from the interviewee's statements (table 11).

TABLE 11 Interview 6 results

Original expression	Category	Failure factor
If a requirement is made in a way that it can be understood in two different ways, the customer will understand it in a way that is better from their perspective.	Quality of requirements	Requirements not specific enough
Transferring of information fails. The customer's actual business needs or processes have not been understood.	Quality of requirements	Unclear requirements

The interviewee emphasized the importance of requirements being precise multiple times during the interview and stated that *requirements not being specific enough* is a common failure factor in software projects. He argued that if a requirement is vague enough to be understood in more than one way, the customer will probably understand it in a way that suits them best, and which is often overly optimistic. This can lead to disappointments on the customer's side, as the project's end product might not fulfil the customer's expectations well enough.

Another recurring failure factor that arose during the interview was *unclear requirements*. According to the interviewee, this failure factor stems from the project team not understanding the customer's business needs or processes clearly enough. Thus, the requirements do not enclose enough accurate information of what the customer really wants from the project.

5.1.7 Interview 7

Interviewee #7 has worked with information technology for 25 years. His job titles have ranged from technical sales support to product manager and project manager, with the main emphasis of his career being on telecommunications field of business. He currently works as a partnerships director for a medium IT corporation. RE failure factors collected from the interview are listed in table 12 below.

TABLE 12 Interview 7 results

Original expression	Category	Failure factor
The requirements often explode during projects. People think that new things can be done within existing projects with little effort.	Requirements management	Changing requirements
Correct representatives from every relevant stakeholder are not present in requirements development. An executive is not necessarily the correct representative.	Requirements development	Wrong people carrying out requirements development
Requirements are left incomplete and superficial, which leads to bad general view of the project and dependencies being overlooked.	Quality of requirements	Requirements not specific enough

Changing requirements was the first recurring failure factor mentioned by the interviewee. He stated that requirements often "explode" during projects, which means that new requirements are added, or existing requirements are modified lightly. Stakeholders seem to think that minor changes can be "smuggled" along with existing requirements, and they tend to think that this can be done without any major impacts to overall cost or schedule of the project. The interviewee emphasized that when changing requirements causes problems, the project manager is at fault by letting it happen.

The interviewee argued that often project stakeholders do not have correct representatives taking part in requirements development. For example, an executive might not have enough technical knowledge to be the sole representative of a stakeholder. The interviewee also described a "loud group of people" from the customer's side that regularly wants to take part in requirements development and whose priority number one is that as little changes are made as possible. This group of people can be seen as a manifestation of resistance to change, and a strong project manager is needed to be able to negotiate quality requirements with them. To summarise this description, there are often *wrong people carrying out requirements development*.

The last failure factor that occurred in the interview was *requirements not specific enough*. The interviewee elaborated that this causes problems with budgeting and scheduling of the project, as well as with understanding dependencies that the developed software might have with other systems. According to the interviewee, some dependencies often come as a surprise to the project team after the project has progressed for a long time, which could have been avoided by composing requirements more thoroughly. The worst-case scenario is that

some other system stops working because of dependencies that were not recognized in the project.

5.2 Compiled Results

Out of the seven interviews that were carried out during this research, a total of eight unique RE related failure factors were identified. These failure factors are listed in table 13, where it is also shown how many interviews each failure factor appeared in.

TABLE 13 Compiled failure factors and their occurrences in the interviews

Failure factor	Occurrences in interviews
Requirements not specific enough	7
Changing requirements	5
Wrong people carrying out requirements development	5
Missing requirements	2
Unclear requirements	2
Poor requirements management	1
Requirements conducted too late	1
Rushed requirements development	1

The data shows that according to the interviewees the most prevalent failure factor in software projects is "requirements not specific enough", which was mentioned by every interviewee. Closely behind on shared second place are "changing requirements" and "wrong people carrying out requirements development" with five occurrences each.

By dividing the identified failure factors into the same three categories that were used earlier in the study (requirements development, requirements management, and quality of requirements), we can see how the failure factors are diverged between different aspects of RE (table 14).

TABLE 14 Failure factors identified in the interviews divided into categories

Category	Failure factor
Requirements development	Wrong people carrying out requirements development
	Requirements conducted too late
	Rushed requirements development
	Missing requirements
Requirements management	Poor requirements management
	Changing requirements
Quality of requirements	Requirements not specific enough
	Unclear requirements

Four of the identified failure factors concern requirements development (50 %), whereas two are about requirements management (25 %) and two are about

quality of requirements (25 %). When examining how often failure factors from each category occurred in the interviews, the numbers look like following:

- Requirements development: 9 occurrences (37,5 %)
- Requirements management: 6 occurrences (25 %)
- Quality of requirements: 9 occurrences (37,5 %)

6 DISCUSSION

In this study, a literature review and an empirical research were conducted to gain more information regarding RE failure factors. The goal of the study was to answer these two research questions:

- RQ1: What requirements engineering failure factors are mentioned in existing studies addressing software project failure factors?
- RQ2: What are the most recurrent requirements engineering failure factors in software projects according to IT professionals?

The first research question can be answered by the results of the literature review. From the seven scientific papers that were analysed for this study, a total of 17 RE related failure factors were found. These failure factors are listed in table 3.

Interestingly, all failure factors that were identified among the papers are worded differently and only two of them are semantically identical. Failure factors "misinterpretation of requirements" (Goedeke, Mueller & Pankratz, 2017) and "misunderstanding the requirements" (Schmidt et al, 2001) clearly mean the same thing, but an equality sign cannot be drawn between any other two failure factors. It can be argued that "incomplete requirements" (Clancy, 1995) and "missing requirements" (Ebad, 2020) have similar intents, although the first failure factor can be interpreted to mean that requirements exist, but they are incomplete in some way, whereas the second one points towards some requirements missing completely. The examined studies all mentioning different RE failure factors might indicate that there is little steadily established scientific knowledge about the subject. It can be assumed that if a more robust knowledge base existed, the failure factors mentioned in the studies would be more uniform.

As an answer to the second research question, the most recurrent RE failure factors in software projects according to our empirical research are:

- Requirements not specific enough
- Changing requirements
- Wrong people carrying out requirements development

All seven interviewees mentioned the failure factor "requirements not specific enough", whereas “changing requirements” and “wrong people carrying out requirements development” were brought up by five interviewees each. In addition to these three failure factors, five less frequently occurring failure factors were identified. All identified failure factors can be seen in table 13.

“Requirements not specific enough” being indicated by every interviewee as a recurring failure factor underlines its prevalence in software projects, and the interviewees mentioned several effects that this failure factor has regarding software project success. Multiple interviewees shared the opinion that requirements not being specific enough makes it harder to work efficiently towards the target of the project. According to the interviewees, the failure factor’s consequences also include challenges in budgeting and scheduling of the project and difficulties in accurately fulfilling the customer’s expectations.

According to the interviewees, “changing requirements” has a more straightforward impact on software projects, as almost every interviewee that mentioned the failure factor stated that changing requirements mostly causes additional work from the project team. Surplus work can naturally delay the project at hand, as well as cause additional monetary costs for the project.

When it comes to “wrong people carrying out requirements development”, the interviewees mentioned this failure factor indirectly causing similar repercussions as the two previous failure factors. Unlike the two other failure factors, this failure factor concerns the requirements development process rather than the quality of requirements. Thus, this failure factor often leads to the emergence of other failure factors, for example requirements not being specific enough.

Four of the eight failure factors identified in our empirical study are mentioned in the scientific papers that were examined in our literature review, whereas four are not. The four failure factors that are mentioned in the examined papers are:

- Changing requirements
- Missing requirements
- Poor requirements management
- Unclear requirements

As some of the answers of our empirical study’s interviewees could not be matched with any failure factors found in the examined literature, these four failure factors had to be conducted for the study:

- Requirements conducted too late
- Requirements not specific enough
- Rushed requirements development
- Wrong people carrying out requirements development

It is interesting that as many as four failure factors are not mentioned in the examined literature, especially when considering that “requirements not specific enough” and “wrong people carrying out requirements development” were among the three most frequently occurring failure factors identified in our empirical research. Someone could argue that the newly conducted failure factor

"requirements not specific enough" has a similar meaning with "unclear requirements", which was found in existing literature. However, requirements being unspecific and them being unclear are not the same: when requirements are unspecific, there might exist completely clear requirements which are too broad. On the other hand, requirements might be very specific but written in an ambiguous fashion, resulting in the requirements being unclear and difficult to understand.

The fact that half of the failure factors that were identified in our empirical study are not mentioned in the examined literature further indicates that RE failure factors have not yet been studied comprehensively. If a solid amount of scientific groundwork about the subject existed, a higher proportion of identified failure factors would presumably have been found in existing scientific papers. Table 15 includes a compilation of all failure factors identified in our literature review and empirical research.

TABLE 15 Requirements engineering failure factors identified in existing literature and in the empirical research

Failure factor	**Found in existing literature**	**Found in the empirical research**
Changing requirements	x	x
Conflicting system requirements	x	
Customers/users did not make enough time for requirements gathering	x	
Ill-defined project requirements	x	
Incomplete requirements	x	
Incomplete requirements analysis	x	
Lack of frozen requirements	x	
Misinterpretation of requirements	x	
Missing requirements	x	x
Misunderstanding the requirements	x	
No method for requirements gathering	x	
No single requirements repository for requirements	x	
Poor requirements management	x	x
Project did not have good requirements at any stage	x	
Project not replanned after requirements changes	x	
Requirements conducted too late		x
Requirements not specific enough		x
Rushed requirements development		x
Unclear requirements	x	x
Unprioritized requirements	x	
Wrong people carrying out requirements development		x

6.1 Theoretical and Practical Implications

Software project failure factors have been studied quite extensively in the past, but RE has not been a common focal point of scientific papers. This study attempted to fill this research gap by providing insight of RE related failure factors. The study introduced several failure factors not mentioned in existing literature. These failure factors could be used as a partial baseline in studies that examine failure factors and their rates of incidence in software projects. For example, the newly identified RE failure factors could be included on a list of failure factors that are investigated in a failure factor study.

Another theoretical implication of this study is that there probably exists failure factors that concern RE that have not been identified in existing failure factor studies. Multiple failure factors that were mentioned in our empirical study are not found in existing literature, which hints that more work is needed to establish a rigid foundation of RE failure factors.

In a practical sense, this study gives project managers and other people working with RE information about what are the most common RE related pitfalls in software projects. Being aware of RE's prevailing risks might help avoiding them in real life scenarios. As RE plays a crucial role in software projects and affects all other project activities, it is probably worthwhile to recognize what can go wrong with it in order to increase the success rate of projects and to save time and money.

6.2 Limitations

One limitation of the study concerns partial homogeneity of the interviewees, as they all work for Finnish IT organizations. Even though the interviewees represent multiple different professional titles and have varying professional experience (ranging from 3 years to 26 years), the fact that they are all positioned in Finnish organizations might skew the study's results. Interviewing people from organizations based in other countries could have resulted in different failure factors to be discovered in the study.

It is also worth noting that semi-structured interviews give some freedom to the interviewer to guide the interview process, which affects the reliability of the study. What this means is that different interviewers could theoretically get different results from the same interviewees. The interviewees would probably mention failure factors that they consider to be the most significant to any interviewer, but less impactful failure factors could remain unmentioned to some interviewers.

Another limitation of the study is that some of the scientific papers referenced are quite old. It was challenging to find more than few recent papers concerning RE related failure factors, so there are a couple of papers included from the beginning of 2000's and one from 1990's. The main limitation here is that the

prevalence of different failure factors in software projects might be different today compared to what it was about twenty years ago. However, the main reason to inspect existing literature was more to find a comprehensive list of existing failure factors rather than to find out how frequently each failure factor occurs.

6.3 Future Research

Several new RE related failure factors arose in this qualitative study, but unidentified failure factors probably still exist. Considering the importance of RE in terms of software project success, additional explorative research in the area would be valuable. Also, the impact and incidence rate of already identified RE failure factors could be studied in a quantitative fashion to further increase the understanding regarding the significance and prevalence of each RE failure factor. In other words, it would be valuable to increase the amount of existing information regarding what RE failure factors have the most significant impact to software project success, and how often each failure factor appears in software projects.

Failure factor studies that were examined in our literature review all mention different RE failure factors, which hints that there might not be a broad amount of well-established knowledge about what the most significant RE failure factors are. Software project failure factors in general have been studied comprehensively before, but more research is needed to expand the knowledge foundation regarding RE failure factors. Broadening this knowledge would most likely help failure factor research in general by giving academics a better understanding about what RE failure factors should be included in future studies.

7 CONCLUSION

The aim of this study was to identify the most prominent RE failure factors in software projects. Past research papers have looked into software project failure factors in general quite extensively, but RE has not been the primary focus of many of the papers. This study attempts to broaden the information available about RE failure factors in software projects by providing insight about the subject in terms of existing literature and opinions of experts of the field.

A total of 17 RE failure factors were identified in the literature review of the study, and these failure factors were observed to concern three aspects of RE: requirements development, requirements management, and quality of requirements. Little over half of the identified failure factors are related to quality of requirements, whereas the rest are evenly distributed between requirements development and requirements management.

The empirical research that was conducted in the study yielded a total of eight identified RE failure factors. Four of these failure factors were also found in the literature review. Additionally, four failure factors could not be found in existing literature and had to be conducted for this study. The failure factors that were most frequently mentioned by the interviewees of the empirical research were “requirements not specific enough”, “changing requirements” and “wrong people carrying our requirements engineering”.

REFERENCES

Bass, L., Bergey, J., Clements, P., Merson, P., Ozkaya, I., & Sangwan, R. (2006). *A comparison of requirements specification methods from a software architecture perspective*. Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... & Kern, J. (2001). Manifesto for agile software development.

Bogdan, R., & Biklen, S. K. (1997). *Qualitative research for education*. Boston, MA: Allyn & Bacon.

Cao, L., & Ramesh, B. (2008). Agile requirements engineering practices: An empirical study. *IEEE software*, *25*(1), 60-67.

Cavanagh, S. (1997). Content analysis: concepts, methods and applications. *Nurse researcher*, *4*(3), 5-16.

Charette, R. N. (2005). Why software fails. *IEEE spectrum*, *42*(9), 36.

Chow, T., & Cao, D. B. (2008). A survey study of critical success factors in agile software projects. *Journal of systems and software*, *81*(6), 961-971.

Chung, L., & do Prado Leite, J. C. S. (2009). On non-functional requirements in software engineering. In *Conceptual modeling: Foundations and applications* (pp. 363-379). Springer, Berlin, Heidelberg.

Clancy, T. (1995). The chaos report. *The Standish Group*.

Cockburn, A., & Highsmith, J. (2001). Agile software development: The people factor. *Computer*, *34*(11), 131-133.

Cohen, D., & Crabtree, B. (2006). Qualitative research guidelines project.

De Lucia, A., & Qusef, A. (2010). Requirements engineering in agile software development. *Journal of emerging technologies in web intelligence*, *2*(3), 212-220.

Dey, I. (2003). *Qualitative data analysis: A user friendly guide for social scientists*. Routledge.

Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development.

Ebad, S. A. (2020). Healthcare software design and implementation—A project failure case. *Software: Practice and Experience*, *50*(7), 1258-1276.

Elo, S., & Kyngäs, H. (2008). The qualitative content analysis process. *Journal of advanced nursing*, *62*(1), 107-115.

España, S., Condori-Fernandez, N., González, A., & Pastor, Ó. (2010). An empirical comparative evaluation of requirements engineering methods. *Journal of the Brazilian Computer Society*, *16*(1), 3-19.

Eveleens, J. L., & Verhoef, C. (2009). The rise and fall of the chaos report figures. *IEEE software*, (1), 30-36.

Firesmith, D. (2003). Modern requirements specification. *Journal of Object Technology*, *2*(2), 53-64.

Francis, J. J., Johnston, M., Robertson, C., Glidewell, L., Entwistle, V., Eccles, M. P., & Grimshaw, J. M. (2010). What is an adequate sample size? Operationalising data saturation for theory-based interview studies. *Psychology and health*, *25*(10), 1229-1245.

Fricker, S. A., Grau, R., & Zwingli, A. (2015). Requirements engineering: best practice. In *Requirements Engineering for Digital Health* (pp. 25-46). Springer, Cham.

Galvin, R. (2015). How many interviews are enough? Do qualitative interviews in building energy consumption research produce reliable knowledge?. *Journal of Building Engineering*, *1*, 2-12.

Glinz, M. (2011). A glossary of requirements engineering terminology. *Standard Glossary of the Certified Professional for Requirements Engineering (CPRE) Studies and Exam, Version*, *1*, 56.

Goedeke, J., Mueller, M., & Pankratz, O. (2017). Uncovering the Causes of Information System Project Failure.

Graneheim, U. H., & Lundman, B. (2004). Qualitative content analysis in nursing research: concepts, procedures and measures to achieve trustworthiness. *Nurse education today*, *24*(2), 105-112.

Hickey, A. M., & Davis, A. M. (2004). A unified model of requirements elicitation. *Journal of management information systems*, *20*(4), 65-84.

Hoffmann, M., Kuhn, N., Weber, M., & Bittner, M. (2004, September). Requirements for requirements management tools. In *Proceedings. 12th IEEE International Requirements Engineering Conference, 2004.* (pp. 301-308). IEEE.

Hughes, D. L., Rana, N. P., & Simintiras, A. C. (2017). The changing landscape of IS project failure: an examination of the key factors. *Journal of Enterprise Information Management*.

Hussain, A., Mkpojiogu, E. O., & Mohmad Kamal, F. (2016). The role of requirements in the success or failure of software projects. *International Review of Management and Marketing*, *6*(S7), 306-311.

Ibraigheeth, M., & Fadzli, S. A. (2019). Core factors for software projects success. *JOIV: International Journal on Informatics Visualization*, *3*(1), 69-74.

IEEE. (1990). IEEE standard glossary of software engineering terminology. IEEE Std 610.12-1990, 1-84.

Inayat, I., Salim, S. S., Marczak, S., Daneva, M., & Shamshirband, S. (2015). A systematic literature review on agile requirements engineering practices and challenges. *Computers in human behavior*, *51*, 915-929.

Jiang, L., Eberlein, A., Far, B. H., & Mousavi, M. (2008). A methodology for the selection of requirements engineering techniques. *Software & Systems Modeling*, *7*(3), 303-328.

Jørgensen, M., & Moløkken, K. (2005). How large are software cost overruns. *A Review of the 1994 CHAOS Report*, 8.

Kamalrudin, M., & Sidek, S. (2015). A review on software requirements validation and consistency management. *International journal of software engineering and its applications*, *9*(10), 39-58.

Krippendorff, K. (1980). Content analysis: An introduction to its methodology. *Sage publications*.

Leech, B. L. (2002). Asking questions: Techniques for semistructured interviews. *PS: Political science and politics*, *35*(4), 665-668.

Longhurst, R. (2003). Semi-structured interviews and focus groups. *Key methods in geography*, *3*(2), 143-156.

Mason, M. (2010, August). Sample size and saturation in PhD studies using qualitative interviews. In *Forum qualitative Sozialforschung/Forum: qualitative social research* (Vol. 11, No. 3).

McManus, J., & Wood-Harper, T. (2007). Understanding the sources of information systems project failure.

Nasir, M. H. N., & Sahibuddin, S. (2011). Critical success factors for software projects: A comparative study. *Scientific research and essays*, *6*(10), 2174-2186.

Nuseibeh, B., & Easterbrook, S. (2000, May). Requirements engineering: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 35-46).

Paetsch, F., Eberlein, A., & Maurer, F. (2003, June). Requirements engineering and agile software development. In *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.* (pp. 308-313). IEEE.

Pandey, D., Suman, U., & Ramani, A. K. (2010, October). An effective requirement engineering process model for software development and requirements management. In *2010 International Conference on Advances in Recent Technologies in Communication and Computing* (pp. 287-291). IEEE.

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

Sauer, C., Gemino, A., & Reich, B. H. (2007). The impact of size and volatility on IT project performance. *Communications of the ACM*, *50*(11), 79-84.

Schmidt, R., Lyytinen, K., Keil, M., & Cule, P. (2001). Identifying software project risks: An international Delphi study. *Journal of management information systems*, *17*(4), 5-36.

Schön, E. M., Thomaschewski, J., & Escalona, M. J. (2017). Agile Requirements Engineering: A systematic literature review. *Computer Standards & Interfaces*, *49*, 79-91.

Shams-Ul-Arif, Q. K., & Gahyyur, S. A. K. (2009). Requirements engineering processes, tools/technologies, & methodologies. *International Journal of Reviews in Computing*, *2*(6), 41-56.

Sillitti, A., & Succi, G. (2005). Requirements engineering for agile methods. In *Engineering and Managing Software Requirements* (pp. 309-326). Springer, Berlin, Heidelberg.

Standish Group. (1994). The chaos report. *The Standish Group*.

Standish Group. (2013). CHAOS manifesto 2013. *The Standish Group International*.

Sudhakar, G. P. (2012). A model of critical success factors for software projects. *Journal of Enterprise Information Management*.

Tsumaki, T., & Tamai, T. (2005). A framework for matching requirements engineering techniques to project characteristics and situation changes. *Situational Requirements Engineering Processes*, 44.

Turk, D., Robert, F., & Rumpe, B. (2005). Assumptions underlying agile software-development processes. *Journal of Database Management (JDM)*, *16*(4), 62-87.

Van Lamsweerde, A. (2009). *Requirements engineering: From system goals to UML models to software* (Vol. 10, p. 34). Chichester, UK: John Wiley & Sons.

Verner, J., Cox, K., Bleistein, S., & Cerpa, N. (2005). Requirements engineering and software project success: an industrial survey in Australia and the US. *Australasian Journal of information systems*, *13*(1).

Verner, J., Sampson, J., & Cerpa, N. (2008, June). What factors lead to software project failure?. In *2008 Second International Conference on Research Challenges in Information Science* (pp. 71-80). IEEE.

Wiegers, K., & Beatty, J. (2013). *Software requirements*. Pearson Education.

Wohlin, C., & Aurum, A. (2015). Towards a decision-making structure for selecting a research design in empirical software engineering. *Empirical Software Engineering*, *20*(6), 1427-1455.

Yousef, A. H., Gamal, A., Warda, A., & Mahmoud, M. (2006, November). Software projects success factors identification using data mining. In *2006 International Conference on Computer Engineering and Systems* (pp. 447-453). IEEE.

Zamudio, L., Aguilar, J. A., Tripp, C., & Misra, S. (2017, July). A requirements engineering techniques review in agile software development methods. In *International Conference on Computational Science and Its Applications* (pp. 683-698). Springer, Cham.

APPENDIX 1 INTERVIEW GUIDE (ENGLISH)

1. Introduction
   - Introduce yourself.
   - Tell about the ongoing study.
   - Inform the interviewee that the interview is confidential, and data collected will be anonymized.
   - Ask the interviewee for a permission for audio recording. Inform the interviewee that the audio recording will be deleted after the study is concluded.

2. Background of the interviewee
   - "What kind of professional background do you have?"
   - "What is your current job title?"
   - "What areas of responsibility do you have related to your job?"
   - "What is your role in software projects?"

3. Requirements engineering in general
   - "In what work-related situations do you work with requirements engineering?"
   - "What requirements engineering techniques have you used in software projects? Examples of requirements engineering techniques are interviews, brainstorming and prototyping."
   - "Think of the last software project you were a part of. How were the project's requirements developed and managed throughout the project?"
   - "How important do you think requirements engineering is in terms of software project success?"

4. Requirements engineering and software project failure
   - "What do you think have been the most frequently occurring mistakes related to requirements engineering in software projects that you have been a part of?"
   - "What challenges have poor requirements caused in software projects that you have been a part of?"
   - Define software project failure to the interviewee:
     "In this research, a software project is considered failed if it exceeded its budget or schedule or if its end product was lacking planned features."
   - "Think of a software project that you were a part of and that you consider was a failure."
     - "What went wrong?"
     - "How was requirements engineering carried out in the project?"

    - "Could requirements engineering have been done better? How?"
- "In your opinion, what are the most important success factors related to requirements engineering?"

5. Conclusion
   - Thank the interviewee for taking part in the interview.
   - Tell the interviewee that they can contact you in case they later have further questions regarding the study or the interview.
   - Conclude the interview.

APPENDIX 2 INTERVIEW GUIDE (FINNISH)

1. Johdanto
   - Esittele itsesi.
   - Kerro meneillään olevasta tutkimuksesta.
   - Kerro, että haastattelu on luottamuksellinen, ja että kaikki kerätty data anonymisoidaan lopulliseen tutkielmaan.
   - Pyydä haastateltavalta lupa haastattelun nauhoittamiseen. Kerro, että äänite poistetaan tutkielman valmistuttua.

2. Haastateltavan taustat
   - ”Millainen ammatillinen tausta sinulla on?”
   - ”Mikä on nykyinen työtehtäväsi?”
   - ”Millaisia velvollisuuksia ja vastuualueita työtehtävääsi kuuluu?”
   - ”Mikä on roolisi ohjelmistoprojekteissa?”

3. Vaatimusmäärittely yleisesti
   - ”Millaisissa työhösi liittyvissä tilanteissa joudut työskentelemään vaatimusmäärittelyn kanssa?”
   - ”Mitä vaatimusmäärittelytekniikoita olet käyttänyt ohjelmistoprojekteissa? Vaatimusmäärittelytekniikoita ovat esimerkiksi haastattelut, aivoriihet ja prototyyppien tekeminen.”
   - ”Muistele viimeisintä ohjelmistoprojektia, jossa olit osallisena. Miten projektin vaatimukset muodostettiin ja miten niitä hallinnoitiin projektin aikana?”
   - ”Kuinka tärkeänä pidät vaatimusmäärittelyä ohjelmistoprojektien onnistumisen kannalta?”

4. Vaatimusmäärittely ja ohjelmistoprojektien epäonnistuminen
   - ”Mitkä ovat olleet useimmiten toistuvia vaatimusmäärittelyyn liittyviä virheitä ohjelmistoprojekteissa, joissa olet ollut osallisena?”
   - ”Millaisia haasteita heikkolaatuiset vaatimukset ovat aiheuttaneet ohjelmistoprojekteissa, joissa olet ollut osallisena?”
   - Määrittele haastateltavalle ohjelmistoprojektin epäonnistuminen: ”Tässä tutkimuksessa ohjelmistoprojekti lasketaan epäonnistuneeksi, jos se on ylittänyt budjettinsa, ei ole pysynyt aikataulussaan tai sen lopputuotteesta on jouduttu karsimaan suunniteltuja ominaisuuksia.”
   - ”Muistele jotakin ohjelmistoprojektia, jossa olit osallisena ja joka mielestäsi epäonnistui.”
     - ”Mikä meni pieleen?”
     - ”Miten vaatimusmäärittely toteutettiin projektin aikana?”
     - ”Olisiko vaatimusmäärittelyn voinut tehdä paremmin? Miten?”

- ”Mitkä ovat mielestäsi tärkeimpiä tekijöitä onnistuneessa vaatimusmäärittelyssä?”

5. Yhteenveto
   - Kiitä haastateltavaa haastatteluun osallistumisesta.
   - Kerro haastateltavalle, että hän voi olla sinuun yhteydessä, jos hänelle tulee myöhemmin mieleen haastatteluun tai tutkimukseen liittyviä kysymyksiä.
   - Päätä haastattelu.