CHALMERS

Connecting functional and feature requirements to support alignment to automated system tests

Master of Science Thesis in Software Engineering

DANIEL JONSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Göteborg, Sweden, June 2015

Connecting functional and feature requirements to support alignment
to automated system tests

Daniel Jonsson

Examiner: Matthias Tichy
Supervisor: Richard Berntsson Svensson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2015

Abstract

Requirements engineering and software testing are two closely related areas of software development, and coordinating their functioning can therefore be highly beneficial. Coordination between the two areas can be achieved by effort adjustments in respective area, and this is referred to as alignment. To achieve, as well as to maintain alignment can however be complicated, and several challenges are associated with alignment. In an industrial context through an empirical study does this research explore ways to improve alignment, as well as challenges to alignment. A test structure for higher-level software testing that builds on the hierarchical relationship between functional and feature requirements is proposed as a way to support alignment. The proposed test structure is evaluated for efficiency through two quasi-experiments, as well as its ability to support alignment during a focus group with practitioners. It is found that the proposed structure through modularisation ease maintenance of software tests, while it at the same time supports alignment between requirements engineering and software testing.

Keywords: Alignment, alignment challenges, alignment practises, test structure, software testing, requirements engineering, functional requirements, feature requirements, modularisation.

Acknowledgements

The author of this thesis would like to thank the supervisor Richard Berntsson Svensson for his major support and dedication, as well as invaluable feedback throughout the thesis and the examiner Matthias Tichy. The author would also like to thank the technical expert Emil Alégroth for his research input and expertise which made this thesis possible. Last but not least, the author would like to devote a special thanks to the case company, its employees, and especially the supervisors from the case company for their time and support throughout the thesis.

Daniel Jonsson, Gothenburg, June 2015

Contents

List of Figures

List of Tables

1 Introduction

Software development processes can be divided into several phases, e.g requirements engineering, software design, implementation, and testing (Royce, 1970), each phase focusing on a specific aspect of software development. Traditionally, requirements engineering is one of the early phases with the purpose to evaluate feasibility, elicit, specify, and validate stakeholder requirements on the software (Sommerville, 2007). Specified requirements from the requirements engineering phase can then support following phases in creation and verification of software. Verification is according to IEEE (2012) defined as: *"The process of providing objective evidence that the system, software, or hardware and its associated products conform to requirements..."*, and can be accomplished by test execution on the software. Test execution to verify the software is carried out during the testing phase, where requirements on the software act as criteria for test success. As the two phases of requirements engineering and software testing are separated, but at the same time have a strong dependency, coordinating their functioning is necessary. To adjust the efforts of respective area to coordinate their functioning and optimise the software development, is by Unterkalmsteiner et al. (2014) defined as alignment.

Aligning the two areas of requirements engineering and software testing is important, and according to a study made by Uusitalo et al. (2008), is the most important reason to increase the information flow about requirements to the testing process. Increased information flow between the two areas can be highly beneficial due to their close relation, and Uusitalo et al. (2008) could conclude that among other things, can increased information flow improve requirements and test quality, increase test coverage, and provide more efficient change management.

Alignment of the two areas is however up till now fairly unexplored (Unterkalmsteiner et al., 2014). To increase information flow is not straight forward, and several challenges in aligning the two areas have been reported, e.g tracing between requirements and test cases, full test coverage, and maintaining alignment when requirements change (Bjarnason, 2014). A great part of previous research in alignment have focused on identifying and reporting alignment challenges faced by companies introducing or extending alignment between requirements engineering and software testing (Bjarnason, 2014. Sabaliauskaitė et al., 2010. Larsson and Borg, 2014. Uusitalo et al, 2008). While alignment challenges have been identified and reported, communicating alignment methods dealing with the challenges have not been considered to the same extent (Unterkalmsteiner et al., 2014). To increase alignment, while dealing with the challenges involved, concrete alignment methods need to be evaluated and communicated to academia. This can then provide the industry with efficient methods for how to coordinate the functioning of requirements engineering and software testing, to optimise software development.

The purpose of this study is to identify challenges with alignment, ways to achieve alignment, as well as propose and evaluate a structure to connect functional and feature requirements to support alignment to automated system tests. A test structure connecting functional and feature requirements is defined, and is in the form of a design research evaluated for efficiency, as well as its ability to support alignment between requirements engineering and software testing. To increase knowledge of alignment challenges, as well as ways to achieve alignment, are semi-structured interviews conducted with practitioners at a case company. The research is performed in an industrial context at a case company to enable the test structure to be evaluated in the appropriate environment, as well as provide the research with real business needs.

The thesis is structured as follows: in the next section, background and related work is presented. In section 3, the research methodology is described, together with threats to validity. Section 4 provides a description of the company under study. In section 5, the test structure is defined and described in more detail. This is followed by the results of the study in section 6, and discussion of the results in section 7. The thesis report ends with a section to conclude the research.

2 Background and related work

This chapter will provide the reader with a short background and introduce the reader to the research topic. The background section is followed by a section to discuss and relate previous related work in the area of research.

2.1 Background

This section will provide a short background to the research, and is divided as follows: First is requirements engineering described, next software testing, and lastly is alignment explained.

2.1.1 Requirements engineering

Software is developed with restrictions and expectations in mind, referred to as requirements. IEEE (1990) defines requirement as; *"A condition or capability needed by a user to solve a problem or achieve an objective", "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"*, and *"A documented representation of a condition or capability"*. As declared in the definitions, requirements support software development to solve problems or achieve objectives, as well as act as a contract to be satisfied before delivery of the software. Requirements are typically classified into categories due to their properties, grouping requirements with the same focus, e.g functional or non-functional requirements (Van Lamsweerde, 2001). The purpose of functional requirements is to restrain and define what a system or product must do (Robertson, Robertson, 1999. Sommerville, 2007). A more formal definition of functional requirement is defined by IEEE (1990) as *"A requirement that specifies a function that a system or system component must be able to perform"*. Non-functional requirements on the other hand focus on restraining and defining system qualities on the system to be developed, e.g performance, security, and usability (Van Lamsweerde, 2001). When functional and non-functional requirements are grouped together to define a logic behaviour as defined by product stakeholders, they are together referred to as a feature requirement (Bosch, 2000). A feature requirement can be seen as a high-level requirement, specified by functional and non-functional requirements to form a a *"logical unit of behaviour"* (Bosch, 2000).

Stakeholder requirements on the software are evaluated for feasibility, elicited, specified, and validated during a software development phase referred to as require-

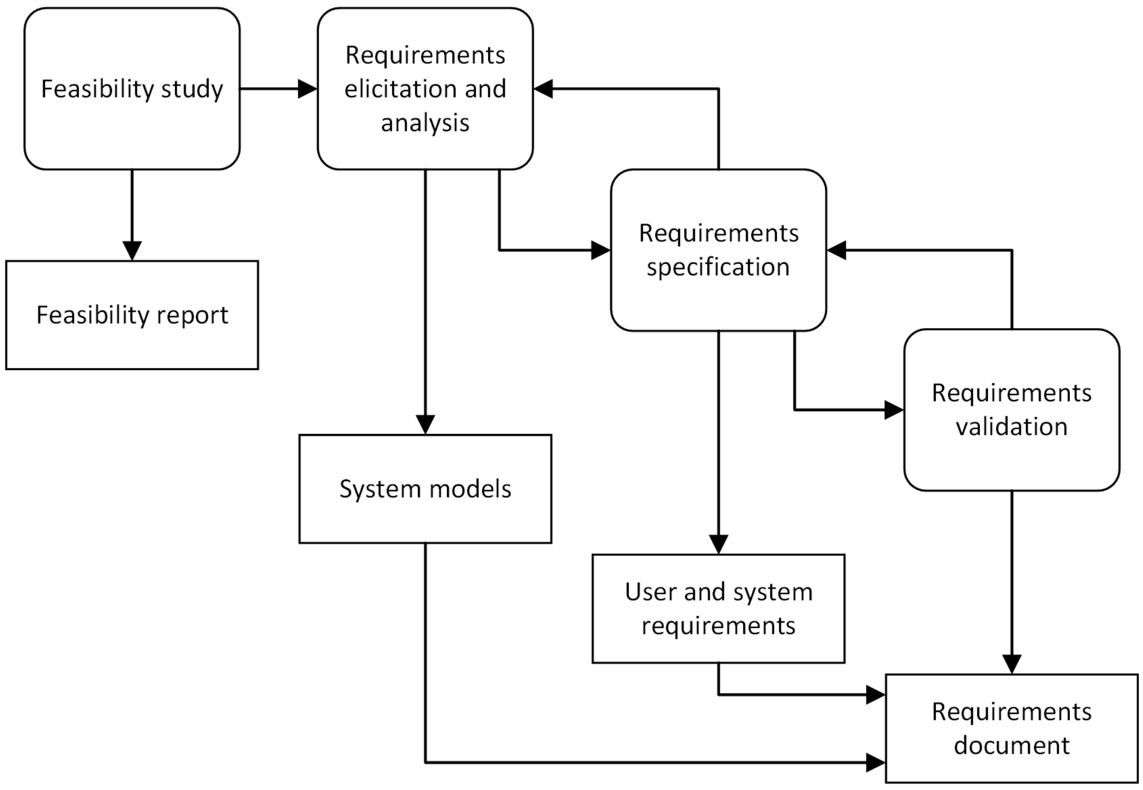


Figure 2.1: RE process model. (Sommerville, 2007)

ments engineering (Sommerville, 2007). As illustrated in Figure 2.1 of an iterative requirements engineering process described in Sommerville (2007), a feasibility study is first performed to evaluate feasibility to develop the proposed system. This means to evaluate and decide if the proposed system will benefit the company objectives, as well as if the project is viable. If the feasibility report is approved, a stage referred to as requirements elicitation and analysis is initiated. Stakeholders, which basically can be any person with an interest in the product, can be involved in requirements elicitation. Requirements are elicited and analysed from stakeholders to acquire conditions and capabilities on the system to be developed. Stakeholder requirements are then specified, to document and communicate the elicited conditions and capabilities. Lastly, to assure that the requirements specified satisfies the stakeholders needs, requirements validation is carried out. The process of requirements engineering is traditionally located at the beginning of the software development process (Royce, 1970), or to better respond to changes in user requirements, incrementally performed during software development, e.g agile software development (Erickson et al., 2005). Resulting requirements from the requirements engineering phase act as guidelines during following design and implementation, and criteria during the verification where the software is verified against its requirements.

2.1.2 Software testing

Stakeholders expect high quality software to be developed, as well as software reflecting the requirements defined on it. Software testing is an activity performed to help ensure this. Software test is defined by IEEE (1990) as *"An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component"*. When a system or component is evaluated against the requirements defined on it, it is referred to as verification. Verification acts as a quality check to ensure that the software under test satisfies the requirements defined on it. To assure software quality is not cheap, according to Rani and Misra (2004), about 50-60% of total project cost is used for testing.

To reduce time spent on manual testing, and reduce the overall costs, automated testing is proposed as one solution (Amannejad et al., 2014). Automated tests reduce the continuous testing effort by reducing the manual effort needed to execute and evaluate the tests. This can be highly beneficial when for example test execution depends on GUI interactions on the software under test. Automatic tests can then be driven through the GUI and evaluated automatically. As automated tests can be executed more often, and requires less manual interaction, it can shorten the feedback loop, and thereby provide the chance to correct problems early after introduction.

During the life-cycle of a software project, code tend to change as an evolutionary process. This can be a result of for example added/removed functionality, corrected bugs, re-factoring etc. As code evolve, not only the modified parts need to be tested, also parts which are dependent on the modified code need to be retested. This type of testing is called regression testing (Srikanth and Cohen, 2011). Regression testing consists of repetitive tasks which are meant to be executed periodically when the code has been modified, to verify that the software still behaves as intended. Regression testing is therefore highly suitable to automate due to its nature of repetitive tasks and periodical execution (Amannejad et al., 2014). Tests can for example automatically be executed during night, and the results evaluated the morning after. Even though automation of regression testing reduces the burden of manual regression testing, the work of maintaining the tests is still present. For regression testing to work as intended, the maintenance of regression tests becomes highly important.

2.1.3 Alignment

In software development, the dependency between requirements engineering and software testing is evident. As visualised in the conceptual V-Model in Figure 2.2 from Sabaliauskaitė et al. (2010), test are used to verify the code, which in turn implements requirements on the software. This means that verification is based on the requirements defined on the software, and verifies that the software correctly reflects its requirements. This means for example that when requirements are added, removed, or modified, the verification criteria is also changed, and therefore the tests associated with the requirements need to be revisited. Due to the strong dependency between the two areas, an arrangement and connection between them can be highly beneficial (Uusitalo et al., 2008). This can be achieved by something called alignment, which is defined by Unterkalmsteiner et al. (2014) as *"The adjustment of RE and ST efforts for coordinated functioning and optimised product development"*.

Alignment is therefore about adjusting the two areas of software development so that they have a coordinated functioning, which shall not be confused with traceability which is about establishing a relationship between the result of two or more areas and defined by IEEE (1990) as *"The degree to which a relationship can be established between two or more products of the development process..."*.

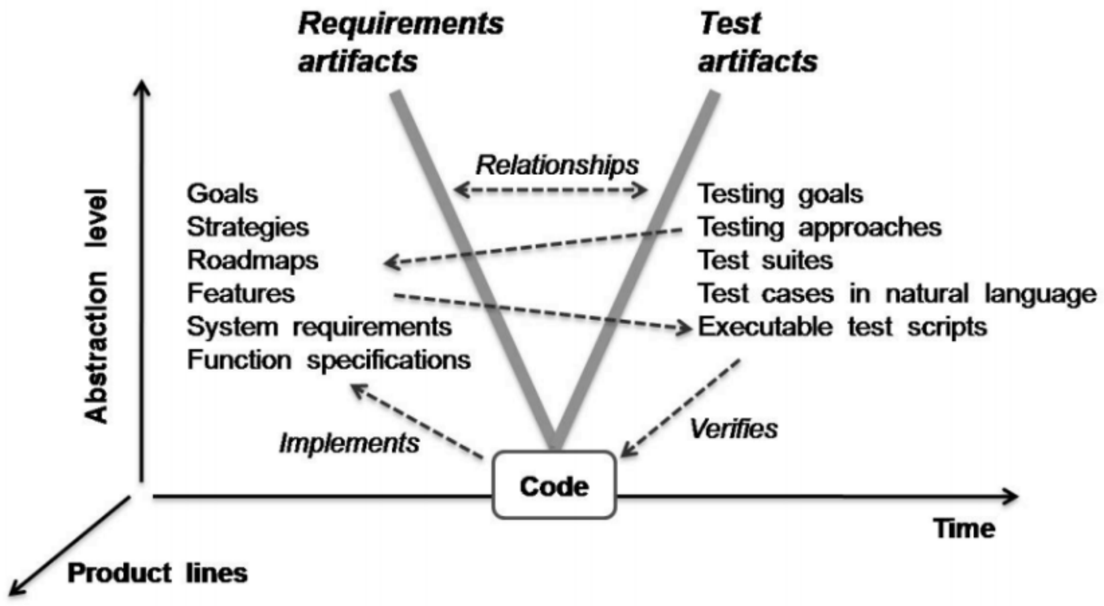


Figure 2.2: Conceptual V-Model. (Sabaliauskaité et al., 2010)

Alignment between requirements engineering and software testing is introduced to achieve certain benefits during the software process. During a series of interviews with five Finnish software companies in 2007, Uusitalo et al. (2008) examined which benefits the studied companies experienced by aligning requirements engineering and software testing. Uusitalo et al. (2008) could conclude that the most important reason for aligning the two areas is to ease and enable information flow about requirements to the testing process. Information flow between the two areas is generated by the introduction of practises into the software development process. Depending on the practise introduced and to what extent, different types of information flow can be achieved, which results in different benefits (Uusitalo et al., 2008). The information flow can benefit both areas separately and embrace quality for both requirements and tests, and thereby increase the overall software quality (Uusitalo et al., 2008).

2.2 Related work

The software engineering fields of requirements engineering and software testing are well explored. Current research has mainly focused on one or the other of the two fields, but the alignment of requirements engineering and software testing is a less explored area (Unterkalmsteiner et al., 2014. Bjarnason et al., 2014). In recent years, some research have been carried out in the area of requirements engineering and software testing alignment. The research has mainly focused on challenges faced by companies trying to increase alignment. Up to this point, a gap in research aiming at understanding and communicating methods aligning requirements engineering and software testing is evident (Unterkalmsteiner et al., 2014). While alignment challenges are identified and discussed, not so much attention has been payed on evaluating a solution to the challenges, which is the main focus of this thesis. The following sub-sections will discuss the different parts of research made in the area of requirements engineering and software testing alignment.

2.2.1 Challenges in RE and ST alignment

The main part of the current research of requirements engineering and software testing alignment deals with exploring and analysing challenges faced by companies that wish to increase alignment between the two fields.

Sabaliauskaitė et al. (2010) conducted an interview study at a large software development company. The objective was to explore current challenges faced by the company when aligning requirements and verification processes. By interviewing 11 professionals at the company on experienced alignment challenges, several challenges hindering requirements engineering and software testing alignment could be revealed. The interviews indicated that software tools are important to the success of alignment, but these are also hindering alignment between the two areas. Both their complexity and lack of common interface makes it hard to achieve and maintain alignment between the two areas. Not only is the lack of communication between tools seen as challenging, also communication and cooperation between different organisational units is seen as problematic. Requirements engineers and testers are possessing different knowledge, and are working at the two ends of the software development process, which makes communication essential. Communication is seen as problematic, especially when facing short deadlines where communication is not given a high priority. This study only included one company, which made it hard to generalise the findings. Therefore, the team extended the study to include more companies and create a broader base for their findings.

Bjarnason et al. (2014) made a case study of six companies where they investigated challenges of requirements engineering and verification alignment. In total, Bjarnason et al. (2014) discovered 16 challenges perceived by the six companies. The 16 challenges could further be divided into requirement specification quality, verification and validation quality, requirements abstract level, and traceability. The six most perceived challenges among the six companies were aligning goals and perspectives within an organisation, cooperating successfully, to define clear and verifiable requirements, defining complete requirements, get a full test coverage, and the tracing between requirements and test cases. A drawback with the aforementioned study was that it only included companies active in proprietary software engineering.

A study by Larsson and Borg (2014) tried to bridge this gap and confirm the findings by revisiting the challenges, but instead in the context of the public sector. The study confirmed most of the findings made by Bjarnason et al. (2014). In total, Larsson and Borg (2014) could draw the conclusion that 11 out of the 16 challenges revealed by Bjarnason et al. (2014) was also relevant in the context of the public sector.

2.2.2 Achieving alignment

To achieve alignment, adjustments in requirements engineering and software testing efforts have to take place. Currently, little research has been done to explore and evaluate concrete methods to achieve alignment between requirements engineering and software testing (Unterkalmsteiner et al., 2014). Despite this, some research have been done to explore present good practises used by companies to create a stronger link between the two areas, as well as practises found in literature which can be suitable for aligning the two areas.

A study made by Uusitalo et al. (2008) presents a set of good practises to align requirements engineering and testing. The presented practises were based on a series of interviews conducted at five Finnish companies. The interviews revealed five practises applied more or less by the different companies. Most of the practises were focusing on employee participation in different stages of the projects, and on peer to peer communication within the organisations. Even though most of the practises considered people relations and communication, the interviews stated that links between documentation is essential for successful testing. The study concluded that linking people, and linking documents in support of each other is more important than of focusing one or the the other to increase software quality.

The study by Bjarnason et al. (2014), which was mentioned earlier on the occasion of challenges to alignment, also explored ways to achieve alignment. Interviewees of the six case companies mentioned in total 27 different alignment practises which were either applied at the companies, or suggested as ways to achieve alignment. The most reported alignment practises mention in Bjarnason et al. (2014), which were mentioned by five out of six companies are the following: Customer communication at all requirements levels and phases, cross-role requirements reviews, process for requirements changes involving VV, tool support for requirements and testing, tool support for requirements-test case tracing. The study revealed that practises connected to tools, change, and requirements are the most commonly applied at the studied companies.

A study by Kukkanen et al. (2009) reports on lessons learned from process improvements by a Finnish company called Metso. To increase work quality did Metso carry out a one year long R&D development project, with the aim to improve the two processes of requirements engineering and software testing. They found that creation of requirements engineering and software testing processes are advantageously carried out simultaneously, as it ease alignment. When the two processes are created simultaneously it becomes easier to adjust their efforts for coordinated functioning, linking requirements and testing etc. The importance of linking people between the two areas was also discovered, more specifically to link requirements- and test managers. By defining clear roles to manage each process will not only help to ensure that information flows between the two areas, it will also help to ensure that the right information flows between the two areas. Change management, and traceability with the help from tools were also found important for linking the two areas.

In addition to the previously mentioned research with the aim to explore present good practises used by companies, have some research been carried out to explore alignment methods found in literature.

Unterkalmsteiner et al. (2014) introduced a requirements engineering and software test (REST) taxonomy to categorise and structure alignment methods found in literature. In total, 635 publications were reviewed, but after application of criteria declaring that publications considered for the REST taxonomy shall consider both areas in the presented method, 630 publications were excluded, and only five alignment methods were found. Among the found alignment methods, a mechanism to create connections between the two areas was most frequently applied.

Barmi et al. (2011) carried out a systematic mapping study focusing on practises which can possibly be used to align requirements specification and testing. The primary focus of the study was to discover alignment practises related to quality requirements, rather than functional requirements. On the other hand, as previously mentioned is there a gap in research related to alignment between the two areas, resulting in a very low number of studies found in total, which is why the study was extended to also include functional requirements. Out of the found papers was model based testing the most frequently found, representing 26% of the papers, followed by formal approaches on 25%, traceability on 18%, and test case generation on 11%.

3 Research methodology

This section will describe how the research is carried out, as well as present threats to validity on the research. First, the research methodology is presented where it is described how the research is prepared, carried out, as well as how the collected data is analysed. The chapter ends with a discussion on possible threats to validity to the research.

3.1 Design research

To fulfil the purpose of the study, four different research questions are established:

- RQ-1: How can alignment be improved and which are the challenges?
- RQ-2: How does the proposed structure support alignment?
- RQ-3: How is the initial testing cost affected by the proposed structure?
- RQ-4: How is the maintenance cost affected by the proposed structure?

During a 5 month period, alignment between requirements engineering and software testing was studied, as well as a structure to achieve alignment created, and evaluated to fulfil the purpose of this study. Design research has been chosen as research methodology for this research, which according to Hevner et al. (2004), is appropriate when then goal of the research is to build and evaluate an artifact with the aim to meet specific business needs. With respect to the purpose of this study, action research has not been chosen as research methodology as its aim is to improve current practice (Robson, 2002), while the purpose of this study is to create and evaluate a new practice. Neither has case study been chosen as research methodology as it according to Runeson and Höst (2009), is merely observational and is used to study contemporary phenomena.

As visualised in Figure 3.1 (Hevner et al., 2004), the general idea of the design research methodology is to make use of existing applicable knowledge, fetched from an already existing knowledge base, to develop or build theories or artifacts. The theories or artifacts are to be evaluated, or justified in the appropriate environment, to solve perceived business needs in the environment. In the end, additions are made to the knowledge base by communicating back the research findings.

In the following sub-sections, we describe the different parts of the design research, and how these are used to answer the four aforementioned research questions. First is the knowledge base described, secondly the environment, and lastly the IS research.

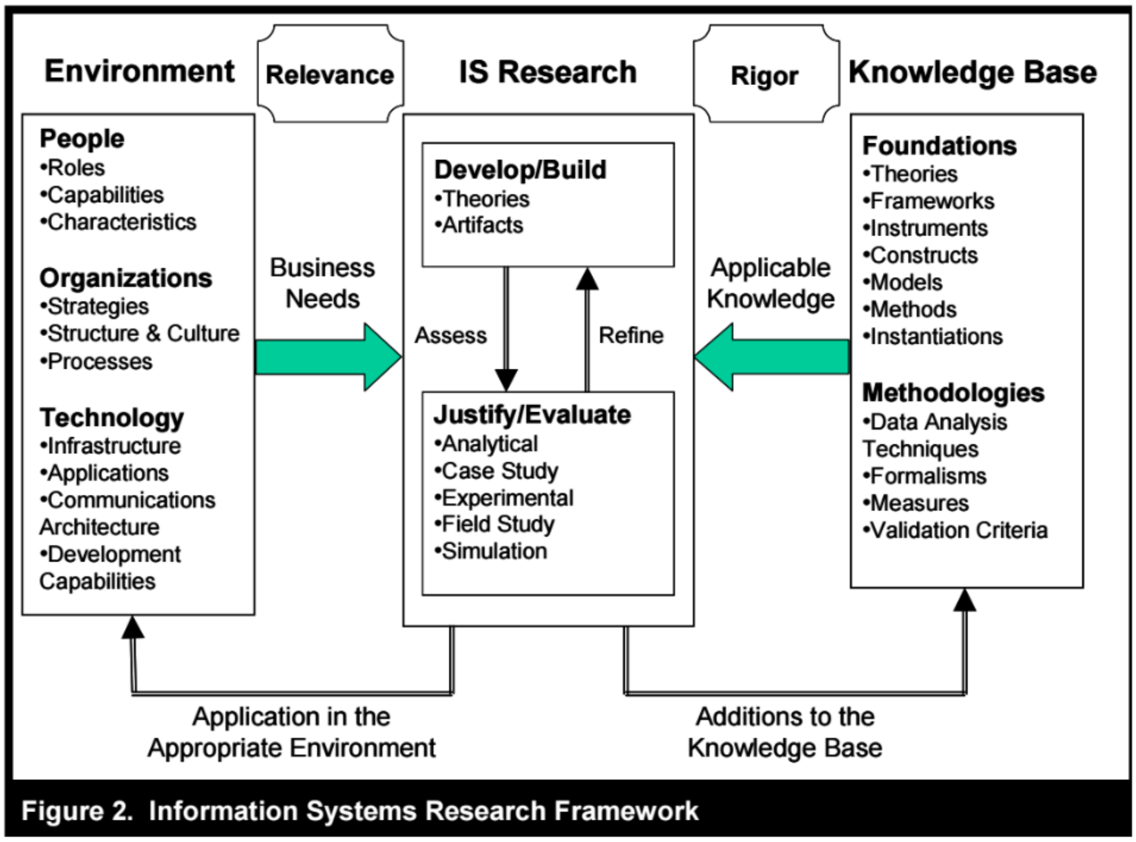


Figure 3.1: Illustration of the design research methodology. (Hevner et al., 2004)

3.1.1 Knowledge base

Applicable knowledge is collected from different sources, mainly from literature which was described in Section 2, but also from the supervisor and technical expert supporting the research. Previous research in the area is first reviewed to increase knowledge about the area, as well as to discover current gaps in research. Literature is as well consulted to correctly carry out the research. Methodologies on how to conduct research is reviewed and compared to each other to apply the most appropriate methodology for this research. The goal is to in the end, make additions to the knowledge base by communicating back the findings of this research.

3.1.2 Environment

To provide the research with real business needs, as well as application in the appropriate environment, it is decided that the research is to be conducted within an industrial setting at a small-sized Swedish software company, hereinafter referred to as the company. The company will be further described in Section 4. The company is planning to implement automatic regression testing for their main software, and can thereby provide the research with the appropriate technology needed conduct the research. People and organisational strategies, structure, culture, and processes within the company provide the research with real business needs, and the opportunity to evaluate the structure appropriately.

Planning: Business needs are collected during a series of semi-structured interviews with practitioners at the company. In the context of alignment between requirements engineering and software testing, are the interviews focusing on suggestions for how to improve alignment and expected benefits to be achieved by it, alignment challenges, as well as current alignment practises. As aforementioned in Section 2.2 about related work, main part of current research in requirements engineering and software testing alignment is focused on the challenges faced when aligning the two areas. Bjarnason et al. (2014) had their interview material created, reviewed, and refined several times by multiple researchers. Considering their focus, as well as the meticulous review and refinement of interview material, it gives us the opportunity to use it as a base for the interview material created to explore business needs in this study (Appendix A). The interview material created by Bjarnason et al. (2014) is published by Runeson et al. (2012, appendix C). To limit the research scope due to time constraints, it is decided that the research should not focus on alignment connected to quality requirements. As quality requirements restrain and define system qualities of the system to be developed (Van Lamsweerde, 2001), they have the tendency to affect other quality requirements, as well as components of the software. To include them in this research would therefore be too problematic due to their interdependencies (Karlsson et al., 2007). By this reason, it is decided that this research should only focus on functional and feature requirements. The interview guide is therefore only based on general questions in the area of research and questions connected to functional and feature requirements from Bjarnason et al. (2014). Before the interviews are conducted, a final preparation is carried out in the form of a pilot study. A pilot study is according to Yin (2003) important as it can support in refining the data collection plan, as well as assist in the creation of relevant questions. The pilot study includes the supervisor of this research, contributing with valuable feedback on both the data collection plan, as well as on the interview questions. The interviews are designed to take approximately 30 minutes, and the resulting interview guide is appended to Appendix A. The selection of participants for the interviews is performed in two ways; company analysis of appropriate participants, and snowballing at the end of each interview. Company analysis is carried out by talking to employees at the case company, to discover different roles who can be interesting to talk to. Snowballing is carried out at the end of each interview, where the interviewee is questioned about other important roles to talk to. In the end, seven different practitioners who are described in Table 3.1, are interviewed at the case company.

Data collection: The interviews are conducted as semi-structured interviews, following the interview guide in Appendix A. Semi-structured interviews does, in contrast to structured interviews, allow improvisation and exploration during the interviews (Runeson and Höst, 2009). The typical focus of semi-structured interviews is to collect qualitatively and quantitatively data on how individuals experience a phenomenon (Runeson and Höst, 2009). As the focus is to collect qualitatively data on how practitioners experience alignment, i.e current business needs, semi-structured interview is found suitable to achieve the goal. Questions for semi-structured interviews are planned and prepared in advance, but the order in which the questions are asked, does not strictly have to follow the interview guide (Rune-

ID	Role
1	Software Developer
2	Software Developer
3	Development Lead
4	Sales Support, Installation
5	Support, Quality Manager, Product Owner, Tester
6	Developer, Tester
7	CTO

Table 3.1: Different roles interviewed at the case company

son and Höst, 2009). The interview guide in Appendix A consists of several steps which constitutes the structured part of the interviews. These steps are asked in the same order as they appeared in the interview guide to reduce influence on subjects. Apart from the aforementioned restriction, interview questions within the same step have no specific order to be asked during the interviews.

Data analysis: Analysis of qualitative data from the semi-structured interviews begins by transcribing the recorded material. The recorded material is listened to, and at the same time written down in a document, divided according to the different interviewees. The transcribed material is then coded, supported by the coding guide provided by Bjarnason et al. (2014), and published by Runeson et al. (2012, appendix D) as a base. The coding guide is also appended to Appendix B. A coding guide is used to in a structured way, analyse the collected data, summarise alignment challenges, current alignment practises, and proposed alignment practises and their benefits as perceived by practitioners, and to be able to present it in a convenient way. Coding is carried out by assigning interview statements different codes, representing categories, as well as sub-categories for each category. Sub-categories can then be analysed to distinguish different aspects of the sub-categories, and be presented in a convenient way. An example of the interviews coding procedure is presented in Figure 3.2. Result of the data analysis on the interviews is presented in section 6, and discussed in section 7.

3.1.3 IS Research

As a solution to some of the business needs discovered during the semi-structured interviews on alignment, a testing structure connecting functional and feature requirements is developed and evaluated. Development of the structure, as well as evaluation of it, is further described below in respective sub-section.

3.1.3.1 Develop structure

As part of the IS research, a test structure in the form of a requirement-test alignment structure is developed. The structure is mainly developed in close collaboration with the technical expert of the research, based on his previous research and knowledge in the area. Feedback on the structure is provided by the supervisor of the research, as well as from the industry where the structure is discussed and

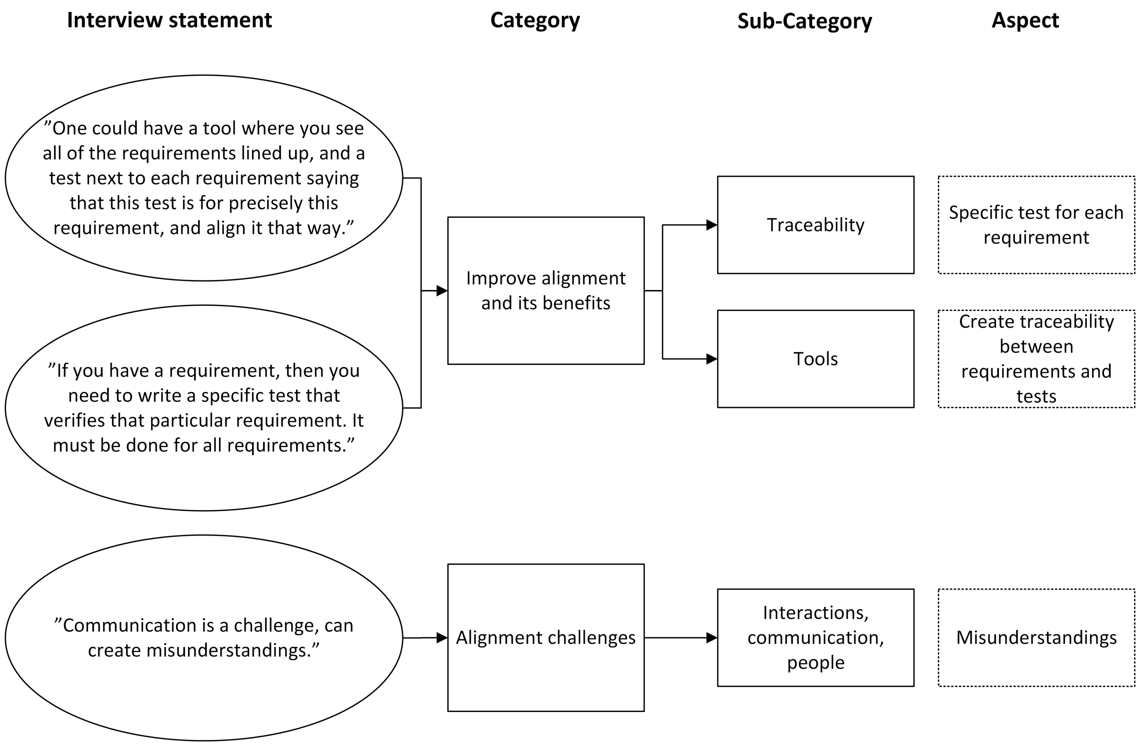

Interview statement
Category
Sub-Category
Aspect
"One could have a tool where you see all of the requirements lined up, and a test next to each requirement saying that this test is for precisely this requirement, and align it that way."
"If you have a requirement, then you need to write a specific test that verifies that particular requirement. It must be done for all requirements."
"Communication is a challenge, can create misunderstandings."
Improve alignment and its benefits
Alignment challenges
Traceability
Tools
Interactions, communication, people
Specific test for each requirement
Create traceability between requirements and tests
Misunderstandings

Figure 3.2: Example of the interview coding procedure.

refined with the help of practitioners in the area. The resulting requirement-test alignment structure is more thoroughly described in section 5.

3.1.3.2 Evaluation

Evaluation of the requirement-test alignment structure is carried out in three different ways, to answer RQ-2, RQ-3, and RQ-4 respectively. To answer RQ-2, a focus group with practitioners is carried out at the company. To answer RQ-3, a quasi-experiment with practitioners is carried out. Lastly, to answer RQ-4, a quasi-experiment including the researcher is carried out. Respective evaluation is further described below, starting with the focus group, followed by the experiment on initial cost, and ends with the experiment on maintenance cost.

Focus group: To answer *RQ-2*, a focus group is carried out with practitioners at the company. The discussions of the focus group are based on the results of the semi-structured interviews on alignment, and concerns the test structure impact on alignment. The discussions of the focus group are divided into four main steps, and discussed in the same order as presented below:

1. General opinions on the proposed structure with its pros and cons.
2. General opinions on the impact of the proposed structure on current alignment practises, followed by opinions on its impact on current alignment practises based on the interview results.
3. General opinions on the impact of the proposed structure on ways to improve

alignment and its benefits, followed by opinions on its impact on ways to improve alignment and its benefits based on the interview results.

4. General opinions on the impact of the proposed structure on alignment challenges, followed by opinions on its impact on alignment challenges based on the interview results.

The focus group is conducted shortly after the quasi-experiment on initial development cost, and includes the same five participants as for the quasi-experiment. This arrangement is used to give the participants of the focus group hands-on experience of the proposed structure before the focus group, and thereby promote discussions. At the beginning of each discussion based on specific interview results, a paper is handed out to the participants with an overview of the specific interview results under discussion. This allow the participants to discuss the topic with respect to the results of the interviews, and at the same time add + or - to the sub-categories they believe would benefit or not benefit from the proposed structure. Data is collected in two different ways during the focus group: discussions are recorded using a voice recorder, and the papers from each participant are collected for further analysis.

Analysis of data from the focus group begins by transcribing the recorded material. The recorded material is transcribed in a similar way as for the interviews, by listening to the recorded material, and at the same time write down the information in a document. The transcribed material is then divided according to the three main categories of the interviews, namely: current alignment practises, improve alignment and its benefits, and alignment challenges. Each category is then divided according to the sub-categories discovered during the interviews for the specific category. The next step is to summarise the papers collected from the participants at the end of the focus group into tables, one table for each category. In each table are the sub-categories for respective category lined up vertically, and id's of the participants horizontally. Final results of each sub-category for each table are then connected to the arguments provided by the participants for the specific sub-category, and the results are presented in chapter 6.2, and discussed in chapter 7.

Experiment initial cost: To answer *RQ-3*, a quasi-experiment is conducted in an industrial context with practitioners at the company. According to Wohlin et al. (2000) are quasi-experiments common in software engineering research, as full randomisation of participants is often hard to achieve. It is infeasible to fully randomise participants for this research, which is the reason why a quasi-experiment is carried out instead of a true experiment. The purpose of the quasi-experiment is to measure how much time it takes for each participant to develop two different test artifacts, verifying the same functionality as common denominator, but which are based on different structures. One of the test artifacts is based on the proposed structure connecting functional and feature requirements, hereinafter referred to as the proposed structure, which is further described in section 5. The other test artifact is based on a more static structure currently used by the case company, where functional and feature requirements are not distinguishable, and modularisation of code is not applied, hereinafter referred to as the old structure. The old structure is further described in section 4. A requirement specification is created before the quasi-experiment is carried out, containing both functional and feature requirements of the test artifacts to be developed. The requirement specification contains one feature requirement, consisting of five functional requirements. A short introduction to CodedUI is also prepared, which goes through the basics of how CodedUI works. The number, as well as complexity of the requirements in the requirement specification are chosen so that the experiment will require approximately 4 hours to be conducted. The ideal case for the experiment would be for each participant to develop two test artifacts of authentic size to cover the whole system. But due to lack of resources, this would not be realistic. Size and complexity of the test artifacts must therefore be adapted to the amount of resources allocated to the experiment by the company. As the company is fairly small, selection of participants for the quasi-experiment is done by including all available software developers and software testers of the company, meaning that five participants are selected for the quasi-experiment.

The experiment is carried out in an industrial context at the case company, where a short introduction to CodedUI is first given to all participants. As the purpose of the experiment is to compare the two different structures, and not how much time it takes to learn how to use CodedUI for different participants, the short introduction to CodedUI is given to reduce the threat of maturation, meaning that participants get positively affected in the sense of increased knowledge throughout the experiment. CodedUI is mainly used to drive the application under test through its user interface, and in that way automate the testing process (Verifying Code by Using UI Automation, 2015). The interface is not the central part under test, what is tested is the underlying model, and the user interface is mainly used to assist in iterating through the application. The two test artifacts are successively developed by each participant with the programming language C#, and using the Visual Studio Premium IDE. The selection of which participants who should start with respective structure is randomised, and it results in that two participants start with the proposed structure, while three participants start with the old structure. Each participant is given an overview of the assigned structure, the requirement specification is handed out, and the stopwatch is started. Time is measured individually for each participant, and when a participant has completed the first test artifact, time is stopped for that participant, written down, and then same procedure is carried out for the second test artifact, based on the next test structure.

Data collected during the experiment on initial development cost is analysed using descriptive statistics, which according to Wohlin et al. (2000) is useful to in an informal way interpret the data. Data is first structured in two different tables, one table for the first developed test artifact, and one table for the second developed test artifact. In the tables is each subject, the structure used by the subject to create the test artifact, and the time to develop the test artifact included. Data of each table are then compared to each other, as well as compared to the data in the other table. Data analysis often begin with descriptive statistics according to Wohlin et al. (2000), and then continue with inferential statistics to draw conclusions based on the collected data. Though, due to the low amount of data points collected is inferential statistics not carried out in this research. Results are presented in section 6, and discussed in section 7.

Experiment maintenance cost: To answer *RQ-4*, a quasi-experiment is conducted in an industrial context at the company. A quasi-experiment is conducted instead of a true experiment for the same reason as for the quasi-experiment on initial cost, as full randomisation of participants is infeasible. To prepare for the quasi-experiment are two different test artifacts developed, verifying the same functionality as common denominator, but which are based on different structures. One of the test artifacts is based on the aforementioned proposed structure, while the other test artifact is based on the aforementioned old structure. Two different features are contained in each test artifact, referred to as FeR1 and FeR2. Each feature contain six functional requirements, where the functional requirements referred to as FR1, FR2, FR4, and FR5 are shared between the two features, and functional requirements referred to as FR3.1 and FR6.1 are specific for FeR1, and functional requirements referred to as FR3.2 and FR6.2 are specific for FeR2. The purpose of the quasi-experiment is to carry out maintenance work on the two test artifacts respectively, and at the same time measure the time required.

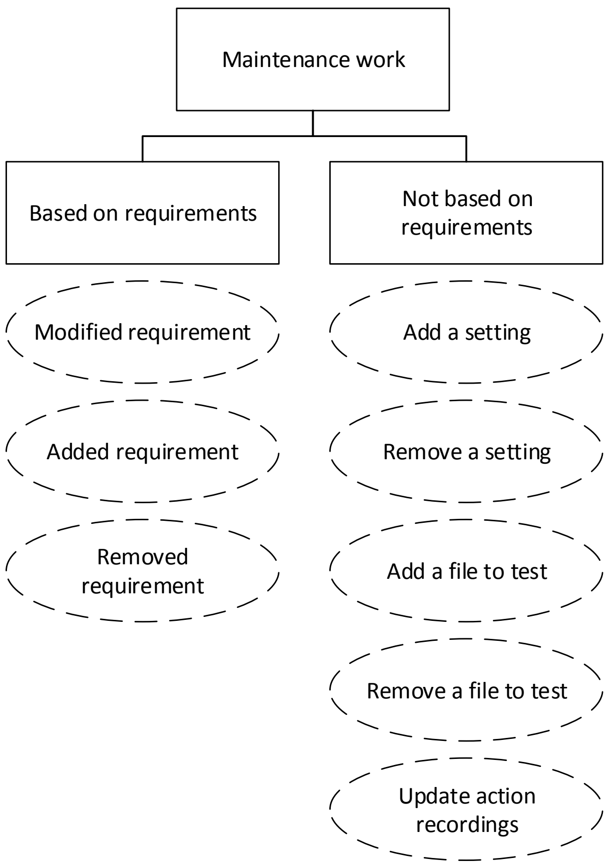


Figure 3.3: Categories of maintenance work.

As visualised in Figure 3.3, two different categories of maintenance work is carried out on the two test artifacts, namely: based on requirements, and not based on requirements. Figure 3.3 also visualises different types of maintenance work for each category, where maintenance work based on requirements includes: modified requirement, added requirement, and removed requirement. Maintenance work not based on requirements includes: add a setting, remove a setting, add a file to test, remove a file to test, and update the action recordings. The five different kinds of maintenance work which are not based on requirements are all selected as they, with respect to discussions with practitioners at the company, are the most probable kinds of maintenance work not connected to requirements of the artifacts. Add or remove settings to test, as well as add or remove files to test are probable maintenance work due to time constraints. To test all different permutations of settings, as well as all different types of files is unreasonable due to the tremendous test execution time it would result in. Therefore, one need to decide on the most reasonable combinations and permutations of settings and test files to execute tests on at the time. As time goes by, the most reasonable combinations and permutations will most probably change, and the tests need to be maintained. Update action recordings is also probable maintenance work needed as time goes by. The action recordings are based on item properties of the GUI. As properties of the GUI items are changed, so must the action recordings of the tests to continue working. Coded UI uses properties of the GUI items as search criteria when iterating through the GUI, meaning that if item properties are changed in the GUI and not updated in the tests, they wont be found, and the test execution will end. As maintenance work on the artifacts require profound knowledge of the artifacts to be maintained, and considering the size of the developed artifacts, resources provided by the company will not suffice to include employees in the experiment. An option would be to carry out maintenance work on the test artifacts developed during the quasi-experiment on initial development cost, but as it is sought to carry out maintenance work on authentic test artifacts, their size will not suffice. Therefore, it is decided that the researcher shall participate in the experiment. This is not optimal for the validity of the research, but because the benefits outweigh the disadvantages, it is decided that so will be it, and that focus on precautions to reduce threats to validity will be taken, which are further described in Section 3.2 about threats to validity.

The experiment is conducted in an industrial context at the case company. Data is collected during the experiment by measuring the time it takes to maintain the two test artifacts, based on the two aforementioned categories. By the reason of random selection, maintenance work is first carried out and measured on the test artifact based on the proposed structure, and secondly on the test artifact based on the old structure for each category. Maintenance work not based on requirements is first carried out. Each kind of maintenance from the category is carried out in the same order as they appear in Figure 3.3, from top to bottom. Maintenance time is measured individually for each kind of maintenance work, for both test artifacts. Next is maintenance work based on requirements carried out. Each functional requirement is first maintained in the test corresponding to FeR2, and secondly the test corresponding to FeR1. Maintenance with respect to functional requirements are made in increasing order, starting with FR1, and ending with FR6. Lastly, FR7 is added, and then removed. Maintenance time is measured individually for each functional requirement in test, for both test artifacts.

Data collected during the quasi-experiment on maintenance cost is analysed in a similar way as for the quasi-experiment on initial cost, using descriptive statistics. Data is first structured in three different tables, one table for the maintenance work not based on requirements, and one table each for the two test artifacts where maintenance work is based on requirements. In the table structuring data from the maintenance work not based on requirements are the five different kinds of maintenance work included, as well as the measured times for both of the artifacts maintained. In the tables structuring data from the maintenance work based on requirements are each modified functional requirement, feature requirement, and the time to maintain respective part associated to the requirements included. Data of the tables are then analysed using descriptive statistics. Data analysis do not continue with inferential statistics for the same reason as for the quasi-experiment on initial cost, i.e due to the low amount of data points collected. Results are presented in section 6, and discussed in section 7.

3.2 Validity threats

There is always an imminent risk that research results are biased by the researchers. To increase trustworthiness of the results, threats to validity need to be considered, and addressed during all phases of the research (Runeson and Höst, 2009). When threats to validity are discussed by academia and in literature, they are generally categorised to distinguish different aspects of validity. As discussed in Berntsson Svensson (2011), these aspects are referred to differently by different authors. Runeson and Höst (2009) as well as Yin (2003) categorises validity by distinguishing four different aspects of validity, namely: construct validity, internal validity, external validity, and reliability. In a similar way in experimental research, Wohlin et al. (2009) categorise validity according to the four aforementioned aspects, with the difference that the fourth aspect, Runeson and Höst (2009) and Yin (2003) decides to call reliability, is by Wohlin et al. (2009) referred to as conclusion validity. As Runeson and Höst (2009), Yin (2003), and Wohlin et al. (2009) address the same aspect with different names, it is more of a matter of taste how to refer to the aspect. In this thesis it is decided to categorise the different aspects of validity in the same manner, referring to the fourth validity aspect according to Wohlin et al. (2009), e.g conclusion validity. This section will define each of the four aspects on validity respectively, as well as describe threats to validity on this study following checklists discussed in Wohlin et al. (2009) for experiments and Yin (2003) on case studies, and how these validity threats have been managed.

3.2.1 Construct validity

This aspect of validity concentrates on the relation between what the researcher intend to measure, and what is actually measured, i.e to what extent does the actual measures represent the intention of the researcher (Runeson and Höst, 2009).
Mono-method bias: There is a risk that measure or observation gives a measurement bias, meaning that if only a single type of measure or observation is used, and it is biased, then the experiment will be misleading (Wohlin et al., 2009). As quasi-experiment is the only method used to measure maintenance and initial costs,

a threat regarding mono-method bias is present. However, as quantitative data is collected, and the quasi-experiments are well structured, mono-method bias is perceived as low. **Hypothesis guessing:** An imminent risk in experiment is that the participants base their behaviour on guesses about the expected outcome (Wohlin et al., 2009). Hypothesis guessing is possible in the quasi-experiments conducted, however as the case company where the quasi-experiments is carried out are looking for best practice regarding treatment, hypothesis guessing is perceived as low. **Experimenter expectancies:** Research results can be biased by the researchers, either consciously or unconsciously based on experimenter expectancies (Wohlin et al., 2009). To reduce this threat, a peer-reviewed interview guide is followed during the semi-structured interviews, presenting the same questions to all participants to reduce the risk of asking questions in different ways among the participants. As the threat considers impact of researcher expectancies on the results, and considering that the researcher got no personal gain in the outcome, nor any expectancies of the outcome as it is a comparison of best practice, threat to experimenter expectancies is seen as low.

3.2.2 Internal validity

This aspect of validity concentrates on the causal relation between factors, i.e is an investigated factor only affected by the factor it is treated with, or is there a possibility that an other factor is causing the effect (Runeson and Höst, 2009). **Maturation:** As a study progress, subjects of the study may react differently, both positively and negatively, which is referred to as maturation (Wohlin et al., 2009). As time passes, subjects may get negatively affected in the sense of for example boredom and tiredness, negatively affecting the results (Wohlin et al., 2009). Subjects may also get positively affected in the sense of for example increased knowledge, which may happen at different rates (Wohlin et al., 2009). To reduce the effect of maturation in this study, quasi-experiments, as well as case studies are kept as short as possible, without affecting the data collection process. Participants of the quasi-experiment on initial cost are also provided with a short introduction to Coded UI before the quasi-experiment, to reduce the effect of increased knowledge of Coded UI throughout the quasi-experiment. **Selection:** Depending on which subjects who are selected for a study from a larger group, they might not be representative for the population (Wohlin et al., 2009). To reduce the effect of selection, a variation of roles at the case company are initially selected for the interviews. At the end of each interview, the interviewee is asked if there are any other relevant roles or people to talk to, called snowballing. Snowballing is carried out to not omit any important role or person, and thereby better represent the population. The selection of participants for the quasi-experiments are perceived as representative for the population as they include both developers and testers, or participants with experience in both areas. The quasi-experiment on maintenance cost includes one participant (the researcher), with experience in both software development and software testing, and have knowledge about the test artifacts to be maintained, as well as their content. This is perceived as representative as practitioners with this background are most probable to carry out maintenance work on similar test artifacts.

3.2.3 External validity

Generalisability of the findings is the focus of external validity, i.e to what extend does the findings of the study represent the population (Runeson and Höst, 2009). **Interaction of selection and treatment** refers to including participants in the study who are not representative for the population (Wohlin et al., 2009), while **Interaction of setting and treatment** refers to using the wrong setting or material for the research, i.e setting and material not representative to industry practice (Wohlin et al., 2009). To reduce the two aforementioned effects, the experimental environment is made as realistic as possible, i.e to use a case company as environment for the test structure to be evaluated in, as well as to provide the research with real business needs. On the other hand, as Wohlin et al. (2009) mentions, reality is not homogeneous, and to cope with this, it is important to describe the environment in which the research is carried out to make it possible for others to evaluate the applicability in a specific environment. This is done in section 4, where the case company is described in more detail, e.g size, roles within the company etc.

3.2.4 Conclusion validity

This category of validity is by Wohlin et al. (2009) referred to as conclusion validity, while by Runeson and Höst (2009) and Yin (2003) it is referred to as reliability. Despite the name, they define the same type of validity. This aspect of validity concentrates on to what extent the study is replicable, and if we can draw the correct conclusions, i.e is the study dependent on the specific researchers, or is it possible for another researcher to conduct the same study and still yield the same results (Runeson and Höst, 2009). **Fishing:** Fishing means that the researchers search for a specific result, and is therefore a threat to conclusion validity since it may influence the result (Wohlin et al., 2009). As maintenance work on two fairly large test artifacts is carried out, and due to the reason that it has to be performed by someone with great insight into the code and structure of the developed artifacts, it has to be carried out by the researcher. As the maintenance work is carried out by a researcher, risk of fishing is present. To reduce the risk of fishing, all clocks are moved out of sight of the researcher to reduce the risk of subconsciously control maintenance work speed depending on the clock. Also, because the researcher does not have anything to gain by fishing, threat to conclusion validity with respect to fishing is perceived as low. **Random irrelevancies in experimental setting:** When an experiment is carried out, environmental elements are present which may disturb the results, e.g noise, interrupts (Wohlin et al., 2009). As maintenance work is carried out in an industrial context, where environmental elements which may disturb the maintenance work are present, actions to reduce the risk has to be put in place. First, to reduce interrupts, maintenance work is carried out during the evening, when most people at the case company have left the company for the day. Secondly, headphones with noise reduction is used to reduce environmental noise during maintenance work. Initial development work is also carried out in an industrial context with environmental elements which may disturb the quasi-experiment. To reduce the risk of disturbance is the quasi-experiment planned and scheduled in advance, and executed during a quiet day.

4
Case company description

The research has been carried out during spring of 2015, in an industrial context at a small-sized Swedish company with around 25 employees. The company is active in the motion capture area, developing both hardware and software connected to motion capture, which are sold on both the national and international markets. To support this, a wide range of professions are present within the company, including support, software and hardware developers, sales, management etc.

Software development within the company is performed by a small group of software developers, practising agile methods, more specifically the scrum methodology. Requirements are therefore incrementally elicited, specified, and prioritised during software development. Requirements are mainly elicited through three different channels, namely: support, sales, and installation staff. Software is continuously improved as an evolutionary process, leading to requirements changes, as well as continuous introduction of new software bugs. Therefore, regression testing their software before each release is seen as profitable to reduce the risk of releasing new software bugs to the customers. Regression testing the software has until recently been done manually before each release. To ease regression testing, and reduce manual interaction during test execution, the company has started to automate the regression testing using CodedUI. CodedUI is a framework used by the company to drive an application through its user interface, and in that way automate the testing. Automatic regression tests are written in C# using CodedUI to automate the testing. Tests are created to verify functionality of the software under test, but are not structured according to, or directly associated to specific functional or feature requirements on the software. Each test is unique in the sense that modularisation is not adopted, meaning that each test and its different parts are implemented in a more or less static manner, verifying some functionality of the software, without reuse of code in the form of modules. This way of structuring tests is in this thesis referred to as the old structure, as it is the old structure compared to the test structure created and evaluated in this research.

5
Test structure

As part of the design research, a test structure is developed and evaluated. The test structure is illustrated in Figure 5.1, and will be further described below. In the first section is the test structure, as well as its different components described. In the second section are expected benefits of the test structure discussed.

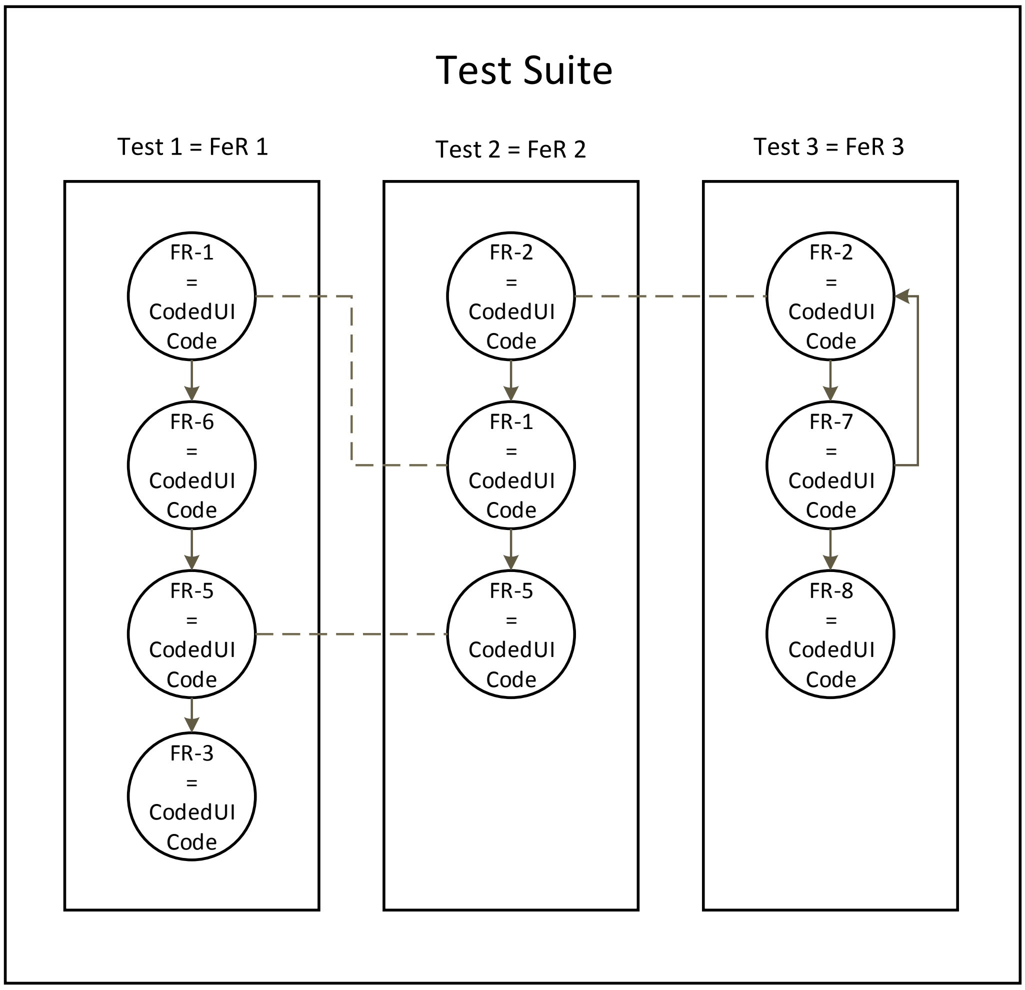


Figure 5.1: Illustration of the test structure.

5.1 Test structure description

The objective of the test structure is to define how software tests, and their content, are to be efficiently structured to support alignment. The test structure builds on the same hierarchical relationship between feature and functional requirements as in the requirements engineering phase. Feature requirements define business value on the software, and can be seen as a high-level requirements, comprised by functional requirements. To align the two areas of requirements engineering and software testing, the test structure adopts the same hierarchical structure for the creation of software tests. Figure 5.1 illustrates the test structure concept, where C# and CodedUI are used to implement tests based on the proposed test structure in this research. Nevertheless is the test structure not restricted to use C# and CodedUI, and any relevant programming language and automation framework can be selected for the cause. The purpose of the test structure is to provide guidelines for how to structure tests and their content, but how these are implemented depends on the specific situation. In this research are C# and CodedUI most suitable as they are already established at the case company, and work well with the software under test.

Test suite can be seen as a container for related tests, and is in Figure 5.1 illustrated as the surrounding square, and referred to as "Test suite". **Tests** are contained in the test suite, and are used to verify the software against its requirements. In Figure 5.1 is each test illustrated as a rectangle within the test suite, and is referred to as "Test". Each test is uniquely associated to a **feature requirement**, which is referred to as "FeR" in Figure 5.1, and the association is illustrated as the "=" sign between a "Test" and "FeR". An association between a test and a feature requirement means that the test uniquely verifies the software against the feature requirement it is associated to. Tests consist of test components in the form of code blocks, in this specific case blocks of **CodedUI code**, illustrated as circles in Figure 5.1, and referred to as "CodedUI code". Each test component is uniquely associated to a **functional requirement**, referred to as "FR" in Figure 5.1, and the association is illustrated as the "=" sign between the "FR" and "CodedUI Code". The meaning of an association between a test component and a functional requirement is that the test component uniquely verifies the software against the functional requirement it is associated to. Each test can consist of one, or many test components, executed in a specific order, illustrated as directed arrows between circles in Figure 5.1. Each test component is strictly defined at a single place, but can be reused by several different tests at the same time. This is illustrated in Figure 5.1 as dotted lines between circles. Test components can for example be defined in methods, which are then called by tests to verify the software against its functional requirements, while at the same time verifying the software against its feature requirements.

5.2 Expected benefits

Each test in a test suite uniquely verifies a feature requirement, and provides a one-to-one mapping between the test, and the feature requirement it is verifying, supporting alignment between the two areas in the form of traceability. In the same manner does each test component uniquely verify a functional requirement to provide a one-to-one mapping between the implementation of the test component, and the functional requirement it is verifying to support traceability. As verified feature requirements, as well as verified functional requirements are clearly distinguishable, awareness of test coverage is perceived to be provided, and thereby alignment supported. Not only is traceability and test coverage expected to be improved, also maintainability is perceived to be increased. As feature tests are comprised of reusable test components of individual functions, maintainability is perceived to be improved due to modularisation. Maintainability is also perceived to be improved due to the isolation of each test component implementation. Each test component is defined at a single place, meaning that changes to a test component only have to be carried out at a single place, and will automatically be propagated to all tests comprised of the test component.

6
Results

This chapter will first provide an overview of current alignment practises at the case company. Next, results of this study divided according to the defined research questions will be presented in respective section, starting with results on how alignment can be improved and expected benefits gained by it, as well as challenges to alignment. Secondly, results on how the proposed structure support alignment is presented. Thirdly, results on how initial testing cost is affected by the proposed structure is presented, and lastly results on how maintenance cost is affected by the proposed structure is presented.

Statements related to how alignment between requirements engineering and software testing is achieved at the case company were categorised as "Current alignment practises" during the data analysis of the interviews. From this main category could five different sub-categories be distinguished. Sub-categories of current alignment practises, as well as alignment practises found for each sub-group are visualised in Figure 6.1, and are further described below.

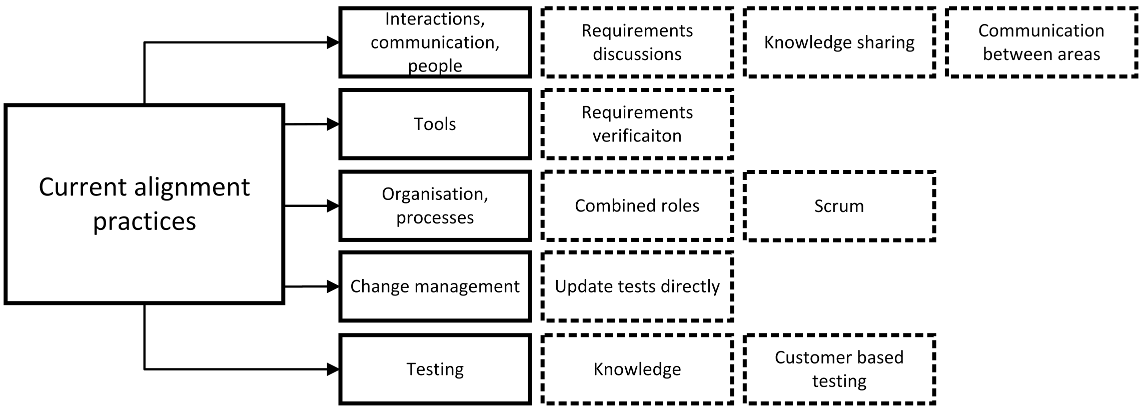


Figure 6.1: Current alignment practises at the case company.

Number of different interviewees who reported alignment practises for respective sub-category is as follows: Interactions, communication, people: 3, Tools: 1, Organisation, processes: 3, Change management: 1, Testing: 3.

Interactions, communication, people categorises statements on how interactions, communication, and people are used to achieve alignment between requirements engineering and software testing. *Requirements discussions:* One interviewee stated that requirements on the software are most often not strictly defined by the customers. A general idea of a feature is usually provided, and requirements of the feature are then discussed among the employees to decide on important requirements to include, prioritise requirements, their testability, analyse cost and benefit etc. *Knowledge sharing:* Knowledge is shared between employees of the company, and most often are those who have been employed the longest the ones who have views on how things should work as they might have a different experience according to one interviewee. Knowledge about for example requirements on the software can then be shared between employees to optimise product development. *Communication between areas:* According to two interviewees is communication between areas what mainly constitutes alignment at the case company. Communication within the team, between the team and product owners, customers etc, makes it possible to share information about respective area with other areas.

Tools categorises statements relating to how tools are used to achieve alignment between requirements engineering and software testing. *Requirements verification:* To ease verification, one interviewee mentioned a tool from Microsoft named Team Foundation Server which is used to increase information flow about requirements to the testing process. Product backlog items are created in Team Foundation Server, which contain information about requirements, and what is to be tested. A formal definition of done is also included in the product backlog items, that somehow puts stamp on what is a finished feature or task. Testers are then able to retrieve information about requirements and definition of done from product backlog items in Team Foundation Server, and use it for verification.

Organisation, processes relates to any interviewee statement about how organisational structure, processes, stakeholders, or roles are used to achieve alignment between requirements engineering and software testing. *Combined roles:* Employees of the studied company often hold multiple roles, and are therefore involved in several areas including requirements engineering, software development, and software testing according to two interviewees. Alignment is implicitly achieved when roles are combined, as information then is shared between areas. *Scrum:* Two interviewees mentioned that the company is practising the scrum methodology, which is a way to support alignment. As an example, after each sprint, a sprint review takes place where completed work is presented to the stakeholders. During the sprint review, stakeholders are given the chance to test the software, as well as provide information about, and verify requirements on the software.

Change management categorises interviewee statements on how change management is used to achieve alignment between requirements engineering and software testing. *Update tests directly:* Requirements can be added, removed, and modified during software development. When this occur, code which is based on the requirements has to be updated accordingly, as well as the tests verifying the code against its requirements. To achieve alignment at the company, one interviewee mentioned that tests are often updated and executed directly after the code has been updated to coordinate the efforts.

Testing includes interviewee statements on how testing efforts are adjusted to achieve alignment between requirement engineering and software testing. *Knowledge:* Testing can be based on knowledge of how customers are using the software, and this type of testing is carried out at the case company according to one interviewee. Employees are using the software as if they were the customers, trying to get the same result as a customer would expect from the software. They use their knowledge of how a customer would use the software to elicit, as well as verify requirements on it. *Customer based testing:* Software testing can be carried out in close collaboration with stakeholders to provide valuable input about requirement to the testing process. Two interviewees at the case company reported that testings is iterated in close collaboration with customers in the form of beta releases, where customers are given the chance to test the software, and the opportunity to provide feedback.

6.1 RQ-1: How can alignment be improved and which are the challenges?

This section will present results of the interviews with practitioners to answer the first research question. First will interview results on how to improve alignment and its benefits be presented, followed by interview results on challenges to alignment.

6.1.1 Improve alignment and its benefits

Statements from the interview material which contained suggestions on ways to improve alignment and possible benefits gained by it were categorised as "Improve alignment and its benefits". Figure 6.2 visualises six different sub-categories distinguished for the category, as well as suggested ways to improve alignment for each sub-category. These are further described below to answer the first part of the research question.

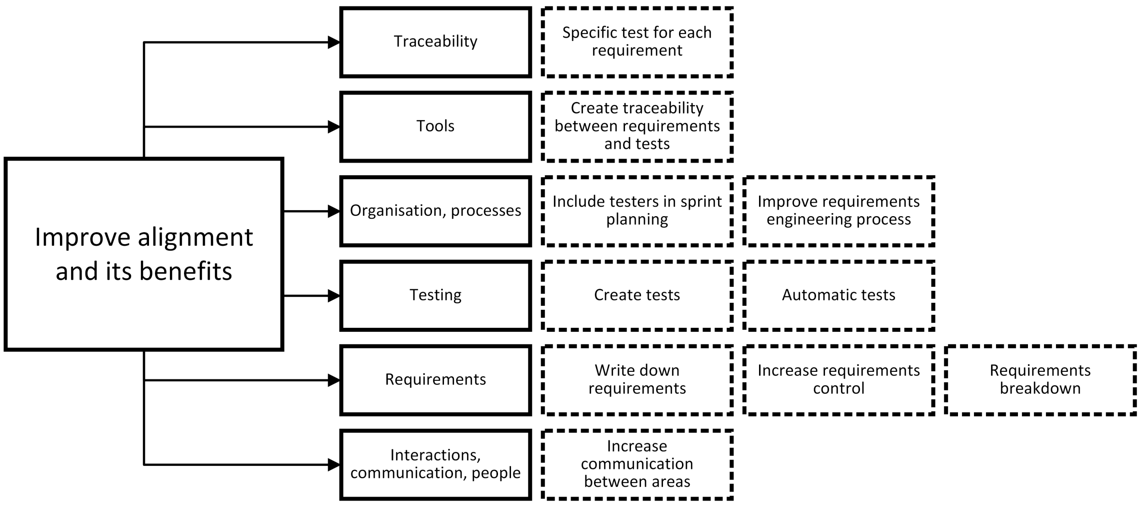

Improve alignment and its benefits
Traceability
Specific test for each requirement
Tools
Create traceability between requirements and tests
Organisation, processes
Include testers in sprint planning
Improve requirements engineering process
Testing
Create tests
Automatic tests
Requirements
Write down requirements
Increase requirements control
Requirements breakdown
Interactions, communication, people
Increase communication between areas

Figure 6.2: Ways to improve alignment reported by the case company.

Number of different interviewees who reported ways to improve alignment for respective sub-category is as follows: Traceability: 2, Tools: 1, Organisation, processes: 2, Testing: 4, Requirements: 4, Interactions, communication, people: 1.

Traceability relates to requirements and testing traceability. Statements on how to improve alignment and its benefits connected to traceability among requirement artifacts and test artifacts, as well as traceability between requirement artifacts and test artifacts, and to code were categorised as traceability. *Specific test for each requirement:* Two interviewees reported that alignment can be improved by introducing traceability between tests and requirements. To achieve this, it was suggested to create a specific test for each requirement, which verifies that the specific requirement is fulfilled. This can and in turn provide an indication of requirements coverage. One of the interviewees stated that:

"If you have a requirement, then you need to write a specific test that verifies that particular requirement. It must be done for all requirements."

Tools relates to tools as a way to improve alignment. Interviewee statements which propose new or present useful tools, or how tools can be used to improve alignment and its benefits were grouped in this sub-category. *Create traceability between requirements and tests:* To improve alignment, one practitioner suggested a tool to create traceability between requirements and tests:

"One could have a tool where you see all of the requirements lined up, and a test next to each requirement saying that this test is for precisely this requirement, and align it that way."

A tool where requirements and tests are aligned to visualise the connection between requirements and tests, i.e to provide information on which requirements are tested, and which tests that verify specific requirements.

Organisation, processes relates to any interviewee statement on how organisational structure, processes, stakeholders, and roles can improve alignment and its benefits. *Include testers in sprint planning:* To make it possible to test the software correctly, one interviewee pointed out the importance that testers are present at the sprint planning. Testers are then given the chance to be informed about requirements, and to provide their opinion and expertise on requirements testability etc. *Improve requirements engineering process:* Improve the requirements engineering process was reported by two interviewees as an important step to improve alignment. By structuring the requirements engineering process in an appropriate way, collecting the right requirements in a clear manner, specifying them at appropriate abstraction levels etc, expected benefits are a better view of the actual requirements on the software so that all requirements can be tested, as well as that the correct requirements are tested.

Testing includes statements about alignment improvements and benefits related to testing. The category does not include statements about change management, processes, and tools as they are separate categories. *Create tests:* Two interviewees pointed out the importance to create tests to verify the software against its requirements to improve alignment. Expected benefits of improved alignment are that with more tests to verify requirements, requirements coverage will increase, meaning that the portion of requirements verified by tests is increased, and that quality of the developed code will increase. *Automatic tests:* The top reported suggestion on how to improve alignment is to introduce automatic tests to verify the software against its requirements. Introduction of automatic tests is reported by three of the interviewees as a good way to improve alignment. Expected benefits with automatic tests are to get rapid feedback about requirements verification, which in turn can provide faster releases with higher confidence on software quality. Also, as automatic tests in a more structured way defines which requirements are tested, requirements coverage as well as continuity are reported to be improved.

Requirements includes statements about alignment improvements and benefits related to requirements. *Write down requirements:* Alignment can according to two interviewees be improved by writing down the requirements of the software, either in a requirements specification or somewhere else. The important thing is not where the requirements are written down, as long as the testing process is provided with concrete and clear requirements to verify the software against. *Increase requirements control:* Two interviewees suggested to increase the requirements control to improve alignment. Requirements control means to have control over which requirements there are on the software, their connections, dependencies etc. Expected benefit of increased requirements control is to better see if the software meets the requirements. *Requirements breakdown:* Before the requirements are written down, they need to be broken down into manageable pieces, and formulated at appropriate level to ease understanding. Breaking down requirements was suggested by one interviewee as a way to improve alignment.

Interactions, communication, people categorises statements related to interactions, communication, and people as ways to achieve alignment and their benefits. *Increase communication between areas:* Even if requirements are written down, communication between areas is reported as an important way to achieve alignment by one interviewee. Increased communication between the ones who elicit the requirements, and those who specify and prioritise the requirements can for example enable requirements to be specified in a correct and clear manner, so that when the software is tested against its requirements, the correct and complete requirements are verified.

6.1.2 Alignment challenges

Statements from the interview material about challenges to maintain, as well as to improve alignment were categorised as "Alignment challenges". Figure 6.3 visualises nine different sub-categories distinguished for the category, as well as challenges for each sub-category. These are further described below to answer the second part of the research question.

Number of different interviewees who reported alignment challenges for respective sub-category is as follows: Interactions, communication, people: 3, Organisation, processes: 3, Testing: 4, Traceability: 3, Requirements: 7, Decisions: 3, Change management: 5, Tools: 1, Resources: 5.

Interactions, communication, people categorises interviewee statements on alignment challenges related to interactions, communication, and people. *Misunderstandings:* Misunderstandings between people, as well as between areas during communications and interactions was reported by one interviewee as an alignment challenge as it can lead to that wrong information flows between people, and areas. *Lack of communication and interaction:* Lack of communication and interaction was reported by two interviewees as a challenge to alignment. This because that

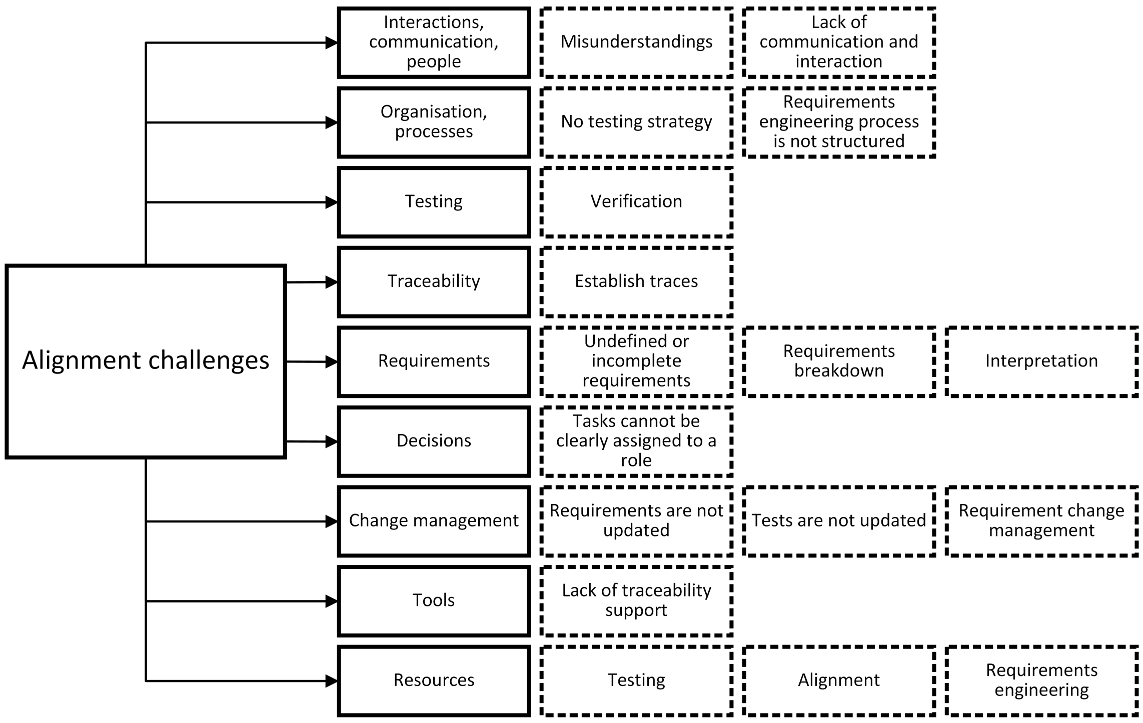

Alignment challenges
Interactions, communication, people
Misunderstandings
Lack of communication and interaction
Organisation, processes
No testing strategy
Requirements engineering process is not structured
Testing
Verification
Traceability
Establish traces
Requirements
Undefined or incomplete requirements
Requirements breakdown
Interpretation
Decisions
Tasks cannot be clearly assigned to a role
Change management
Requirements are not updated
Tests are not updated
Requirement change management
Tools
Lack of traceability support
Resources
Testing
Alignment
Requirements engineering

Figure 6.3: Alignment challenges reported by the case company.

it can lead to reduced information flow between people, and areas. Interactions and communications are presence sensitive, meaning that if for example a particular person does not attend a meeting, information to that person might be reduced if not managed.

Organisation, processes relates to any interviewee statement about how organisational structure, processes, stakeholders, and roles are seen as challenges to alignment. Alignment between requirements engineering and software testing can be a challenge if one or the the other of the two areas is not correctly structured. *No testing strategy:* Two interviewees reported that without a strategy for what, when and how things shall be tested, it is hard to know which requirements that have been verified, and if they have been correctly verified. *Requirements engineering process is not structured:* Also, if requirements are not elicited, specified, prioritised etc in a complete and structured manner, lack of, unclear, or wrong requirements can cause problems for verification of requirements according to two interviewees.

Testing includes interviewee statements on how testing is seen as a challenge to alignment. *Verification:* Verification is performed to ensure that the developed software complies with the requirements defined on it. If testing is flawed, requirements are not verified correctly, meaning that the software might not comply with its requirements, i.e the wrong software has been developed, and will be delivered to the customer. Verification is seen as an alignment challenge by four interviewees, as it can be challenging to verify the software against the correct requirements. Too often, the software is verified against requirements which have evolved during the development, which might not even be close to what the actual requirements on the software are. This means that the software is verified, but the verification is false positive, and the wrong software is delivered to the customer.

Traceability relates to statements about challenges with traceability among requirement artifacts and test artifacts, as well as traceability between requirements artifacts and test artifacts, and to code. *Establish traces:* If traceability is not established between requirements and tests, it is hard to know which requirements are covered by tests, and if they are correctly covered. Three interviewees reported that establishing traces between requirements and tests to see requirements coverage is challenging, mainly due to two reasons: requirements are not written down and stored, and testing is mainly carried out manually. As requirements are not written down and stored somewhere, it is hard to introduce traces to them. Also, if testing is mainly carried out manually by employees, it is hard to introduce traces, making it hard to know if requirements are covered by testing.

Requirements includes statements on alignment challenges and problems related to requirements. *Undefined or incomplete requirements:* Undefined or incomplete requirements was reported as challenging by six interviewees. When requirements are not defined at all, lack information, or are not written down, it becomes challenging to create tests for them, as well as to correctly verify them. *Requirements breakdown:* Three interviewees reported that it is challenging to find an appropriate level to specify requirement on, as well as to break down large requirements into manageable and feasible pieces. If requirements are not broken down enough, it will later be hard to correctly verify them. *Interpretation:* How requirements themselves are interpreted is by different people is a challenge when it comes to alignment according to one interviewee. Requirements can be interpreted in different ways by different people, meaning that different people can have different views on how a software shall behave.

Decisions categorises statements which are related to decision making and distribution. Statements on alignment problems and challenges relating to things that can not be clearly assigned to a role were added to this category. *Tasks cannot be clearly assigned to a role:* Without clear and defined roles, responsibility is shared among people or left out, meaning that there is a risk that tasks are not properly performed, or not performed at all, which is a challenge to alignment according to three interviewees. If there for example is no dedicated tester, there is a risk that requirements are not verified at all. As one interviewee stated it:

"Must have dedicated testers so that it is verified that the requirements are truly met."

Change management categorises interviewee statements on alignment challenges related to how changes are handled. *Requirements are not updated:* Even if requirements are completely and clearly defined, if they are not updated as they change, verifying the requirements becomes a challenge according to four interviewees. *Tests are not updated:* One interviewee stated that if tests are not updated according to the updated requirements, tests are not verifying the correct requirements. *Requirement change management:* If no good process of what will happen when requirements change, it becomes challenging to prioritise the new requirement and develop it according to one interviewee. The interviewee stated that:

"If an important new requirement comes in, it breaks down our structure and we need to prioritise it."

Tools relates to alignment challenges connected to tools. Interviewee statements on challenges with new or present tools, how tools are used, or cannot be used were categorised as tools. *Lack of traceability support:* Tools can often become an alignment challenge as they do not have support for practises to achieve alignment. Traceability between requirements and tests is such a thing which can be problematic when tools are used according to one interviewee.

Resources are grouping interviewee statements which are connected to resources, i.e challenges with resources. Resources are limited, making prioritisation necessary. *Testing:* Three interviewees reported that it is challenging to get testing prioritised. Resources are rather focused on development than testing, meaning that adjustments to software testing efforts are left out. *Alignment:* To prioritise alignment in general is also an alignment challenge according to one interviewee. Resources are often not focused on thinking about alignment and improving alignment. *Requirements engineering:* Similar to testing is requirements engineering not highly prioritised. Resources are focused on development rather than eliciting, specifying, and prioritising requirements according to three interviewees.

6.2 RQ-2: How does the proposed structure support alignment?

This section will answer the second research question related to how the proposed structure support alignment. Findings from the focus group conducted with practitioners are presented to answer the research question. The section is divided into three sub-sections with the purpose to answer three different angles of how the proposed structure support alignment, divided according to the three main categories discovered during the semi-structured interviews. First, findings on how the proposed structure can support current alignment practises at the case company are presented. Next, findings on how the proposed structure can support the suggested ways to improve alignment are presented. Last, findings on how the proposed structure deals with the discovered challenges connected to alignment are presented.

6.2.1 Support on current alignment practises

An overview of the respondents' perception of the structures impact on current alignment practises is summarised in Table 6.1, and is further described below, divided according to the discovered sub-categories of current alignment practises at the case company.

Current alignment practises	**1**	**2**	**3**	**4**	**5**
Interactions, communication, people					+
Tools	+	+	+	+	+
Organisation, processes		+	+	+	+
Change management	+	+	+	+	
Testing	+	+	+	+	+

Table 6.1: Test structures' support on current alignment practises

Interactions, communication, people: According to the participants of the focus group, the proposed structure got no, or uncertain impact on interaction, communication, and people. Only one participant out of five stated that the proposed structure can support the area by the means that it will be a central part of discussions. He stated that:

"It will be part of the discussions, but it is difficult to say if it will have a positive or negative effect because we will not increase communication because of it."

The remainder of the participants argued that it most probably will have an impact on the area, but it is hard to tell if it will be positive or negative.

Tools: All participants agreed on that the proposed structure will support alignment with respect to tools, as the tester more easily can modify affected tests when new product backlog items are created in Team Foundation Server. Product backlog items are closer related to functional requirements, than to specific tests. This means that when a new product backlog item is created, it will be easier for the tester to distinguish functional requirements from the product backlog item, and make changes to the concerned test modules representing those functional requirements, than to distinguish tests to make changes to.

Organisation, processes: Most of the participants (4 out of 5) stated that the proposed structure on software tests will support alignment with respect to organisation and processes. The participants explained that the test structure will make it easier to plan for new sprints during the sprint planning as they can discuss affected tests on module level, i.e which requirements are affected by the changes, and thereby which test modules that need to be modified accordingly, as opposed to which tests that will be affected.

Change management: Alignment with respect to change management at the case company was reported by four out of five participants to be supported by the proposed structure. The participants argued that the proposed structure will raise awareness of what is affected in the tests when requirements are changed. Awareness of which parts that are affected, reduces the risk to have to redo entire tests. One can focus on modules of the tests instead. Also, by the use of modularisation, one can modify at one place, and the modifications will be propagated to all tests based on the module, which in turn will make the tests easier to maintain.

Testing: All participants agreed on that the proposed structure will support alignment connected to testing. It will provide the tester with a clear and structured way for how to create and execute tests to cover all functional requirements of a feature, which provides continuity between test executions, as well as fill out gaps in knowledge about how customers might use the software.

6.2.2 Support ways to improve alignment and its benefits

In this sub-section, results of how the proposed structure will support alignment with respect to the suggested ways to improve alignment and its benefits discovered during the semi-structured interviews are reported. An overview is provided in Table 6.2, and is further described below, divided according to the discovered sub-categories of ways to improve alignment and its benefits.

Improve alignment and its benefits	**1**	**2**	**3**	**4**	**5**
Traceability	+	+	+	+	+
Tools		+	+	+	
Organisation, processes	+	+	+	+	
Testing	+	+	+	+	+
Requirements	+	+		+	+
Interactions, communication, people					

Table 6.2: Test structures' support on improve alignment and its benefits

Traceability: All participants of the focus group reported that traceability will be supported if the proposed structure is used. As each test represents a specific feature requirement, and each module of a test represents a specific functional requirement, the traceability between feature requirements and tests, as well as between functional requirements and modules of tests become evident. It will thereby be easier for the tester to see which tests and modules that need to be modified as requirements change, as well as which requirements are covered by tests. One interviewee said that:

"...using the proposed structure will allow for better traceability and thus better testing."

While an other interviewee stated that:

"It becomes easier to see which changes you need to do in the tests depending on the specific requirements coming in. If you see which changes you need to make in the code, then you also see which changes you need to do in the test, which will be much easier with the proposed structure..."

Tools: Three out of five participants reported that the proposed structure will support tools as a way to improve alignment. One participant argued that instead of only creating traces between requirements and tests with the help from tools, it will be more beneficial to also create traces between requirements and modules. With traces between requirements and modules, it will be easier to understand which parts of the tests that are affected when requirements change, and not only which tests.

Organisation, processes: According to four participants of the focus group, organisation and processes will be supported by the proposed structure. By using the proposed structure for tests, sprint planning for example will be much easier to carry out as requirements and tests will be similar in their representation. Thereby, it will be easier to find out which tests that are affected by the requirements, as well as their testability.

Testing: All participants reported that testing will be supported by the proposed structure. Verification of requirements, as well as requirements coverage will be supported when you have a one to one mapping between requirements and tests, as well a between requirements and modules of tests. With a one to one mapping, it will become clearer which requirements are covered by tests, and it will become easier to verify feature requirements when the functionality is divided, as it will reduce the complexity. As test modules can be reused, the proposed structure should also make it easier and faster to create new tests when requirements are added. The speed will most probably increase over time as the number of modules to select from increases.

Requirements: Most of the participants (4 out of 5) reported that requirements will be supported if the proposed structure is used. Nevertheless, none of the participants could provide an explanation of why the proposed structure will support requirements.

Interactions, communication, people: All participants were uncertain of the structures impact on interaction, communication, and people. As with its impact on current alignment practises, the participants discussed that it most probably will have an impact on the area, but that it is hard to tell if it will be positive or negative.

6.2.3 Impact on alignment challenges

In this sub-section, results of how the proposed structure will support alignment with respect to the alignment challenges discovered during the interviews are reported as perceived by the participants of the conducted focus group. An overview is provided in Table 6.3, and is further described below, divided according to the discovered sub-categories of alignment challenges.

Alignment challenges	**1**	**2**	**3**	**4**	**5**
Interactions, communication, people					
Organisation, processes		+	+		+
Testing	+	+		+	
Traceability	+	+	+	+	+
Requirements		+			
Decisions					
Change management	+	+	+	+	
Tools		+	+		+
Resources		+		+	

Table 6.3: Test structures' support on alignment challenges

Interactions, communication, people: As with previous categories were the participants uncertain of the structures impact on interactions, communication, and people. The participants could not make up their mind on what type of impact the proposed structure will have on the area, even if they were fairly sure that it will.

Organisation, processes: Three of the participants stated that the proposed structure will deal with alignment challenges connected to organisation and processes. The participants discussed that due to its clear and concrete structure, it will be easier to define a test strategy for what, when, and how things shall be tested. If the same structure is adopted during the requirements engineering phase, i.e dividing requirements into feature requirements consisting of functional requirements, also the requirements engineering process will be will be better structured. As one practitioner said:

"It will clearly be improved because it is a very clear and concrete requirements definition process, and it is a very clear way to create requirements where it otherwise

might be difficult to define what the requirement is."

Testing: According to three of the participants of the focus group does the proposed structure deal with alignment challenges connected to testing. Verification of requirements was reported to be easier with the proposed structure as it will provide a one-to-one mapping between requirements and tests. This way, the software will have a higher chance to be verified against the requirements which are defined on the software, and not requirements which have evolved during development.

Traceability: All of the participants stated that the proposed structure will deal with alignment challenges connected to traceability. As the structure provides a one-to-one mapping between requirements and tests, traces between requirements and tests are established. Traces are even established if requirements are not specified in a specification, as the tests in a way specifies the requirements on the software.

Requirements: Most of the participants (4 out of 5) did not know if the structure will have any impact on challenges connected to requirements. As the tests are created after the requirements are defined, it is hard to know if the test structure will have any impact on challenges connected to requirements. On the other hand, one practitioner stated that it will be easier to specify requirements that are hard to specify if you know the process of how it will be verified.

Decisions: None of the participants knew if the proposed structure will have any impact on alignment challenges connected to decisions.

Change management: Nearly all of the participants reported that the proposed structure will deal with the alignment challenges connected to change management. According to the participants are challenges related to change management the most positively affected by the proposed structure. With the proposed structure, test modules only need to be maintained at a single place when a functional requirement is changed, which ease maintenance.

Tools: Three out of five participants stated that the proposed structure will have positive impact on challenges related to tools. As the proposed structure implicitly will provide traceability between requirements and tests, the need for tools to provide traceability is reduced.

Resources: Only two participants stated that the proposed structure will have a positive effect on challenges related to resources. The two participants argued that since the proposed structure divides testing into tests, and modules of tests, resources can be allocated to specific tests or modules of tests, and thereby can the available resources be focused on the most important parts.

6.3 RQ-3: How is the initial development cost affected by the proposed structure?

This section will present the results of the quasi-experiment on initial development cost conducted with practitioners. The individual results are summarised in Table 6.4, and will be further described in respective sub-section below. First are the measurements on the first- and second test artifact presented and compared to each other. Secondly are the measurements on the old structure presented. Thirdly are the measurements on the proposed structure presented. Lastly are the measurements of the two structures compared to each other.

First test artifact			Second test artifact		
Subject	**Structure**	**Time**	**Subject**	**Structure**	**Time**
Practitioner 1	Proposed	81min	Practitioner 1	Old	14min
Practitioner 2	Proposed	78min	Practitioner 2	Old	20min
Practitioner 3	Old	79min	Practitioner 3	Proposed	80min
Practitioner 4	Old	67min	Practitioner 4	Proposed	53min
Practitioner 5	Old	49min	Practitioner 5	Proposed	41min

Table 6.4: Initial development costs measured, divided into first- and second test artifact developed

6.3.1 Comparison between first- and second test artifact

The total time for all participants to complete development of the first test artifact is 354 minutes, compared to 208 minutes for the second test artifact. Mean development time for each participant for the first test artifact is 70,8 minutes, while for the second test artifact the mean development time is 41,6 minutes. Development time for the second test artifact is therefore 59% of the development time for the first test artifact. This means that development of the first test artifact took 1,7 times as long time, compared to the second test artifact. When the first test artifact is based on the proposed structure, and the second test artifact is based on the old structure, development time for the second test artifact is 21% of the development time for the first test artifact. When the first test artifact instead is based on the old structure, and the second test artifact is based on the proposed structure, development time for the second test artifact is 89% of the development time for the first test artifact.

6.3.2 Old structure

First test artifact: Three practitioners developed the first test artifact based on the old structure, and the mean time for the three practitioners is 65 minutes, with a maximum of 79 minutes, and a minimum of 49 minutes.
Second test artifact: Two practitioners developed the second test artifact based on the old structure, and the mean time for the two practitioners is 17 minutes, with a maximum of 20 minutes, and a minimum of 14 minutes.
Comparison of first- and second test artifact: If the mean times of the first- and second test artifact based on the old structure are compared, is the mean time for the second test artifact 26% of the mean time for the first test artifact. This means that the first test artifact took 3,82 times as long time to develop, compared to the second test artifact.

6.3.3 Proposed structure

First test artifact: Two practitioners developed the first test artifact based on the proposed structure, and the mean time for the two practitioners is 79,5 minutes,

with a maximum of 81 minutes, and a minimum of 78 minutes.
Second test artifact: Three practitioners developed the second test artifact based on the proposed structure, and the mean time for the three practitioners is 58 minutes, with a maximum of 80 minutes, and a minimum of 41 minutes.
Comparison of first- and second test artifact: If the mean times of the first- and second test artifact based on the proposed structure are compared, is the mean time for the second test artifact 73% of the mean time for the first test artifact. This means that the first test artifact took 1,37 times as long time to develop, compared to the second test artifact.

6.3.4 Comparison between old- and proposed structure

First test artifact: Mean development time for the test artifact based on the proposed structure is 79,5 minutes, compared to 65 minutes for the test artifact based on the old structure. Mean development time for the test artifact based on the old structure is therefore 82% of the mean development time for test artifact based on the proposed structure. This means that development of the test artifact based on the proposed structure took 1,22 times as long time, compared to the test artifact based on the old structure.
Second test artifact: Mean development time for the test artifact based on the proposed structure is 58 minutes, compared to 17 minutes for the test artifact based on the old structure. Mean development time for the test artifact based on the old structure is therefore 29% of the mean development time for test artifact based on the proposed structure. This means that development of the test artifact based on the proposed structure took 3,41 times as long time, compared to the test artifact based on the old structure.
Total: Total development time for the test artifacts based on the proposed structure is 333 minutes, compare to 229 minutes for the test artifacts based on the old structure. Mean development time for the test artifacts based on the proposed structure is therefore 66,6 minutes, compared to 45,8 minutes for the old structure. Development time for the test artifacts based on the old structure is therefore 69% of the development time for the test artifacts based on the proposed structure. This means that development of the test artifact based on the proposed structure took 1,45 times as long time, compared to the test artifact based on the old structure.

6.4 RQ-4: How is the maintenance cost affected by the proposed structure?

This section will present results of the quasi-experiment on maintenance cost, and is divided according to the category of maintenance work carried out. First, results of the maintenance work not based on requirements are presented, and secondly are results of maintenance work based on requirements presented.

6.4.1 Maintenance work not based on requirements

This sub-section will present results of the quasi-experiment on maintenance cost, more specifically for maintenance work which is not based on requirements. Results of the maintenance work are summarised in Table 6.5.

Maintenance work	Old structure	Proposed structure
Add a setting	280 sec	166 sec
Remove a setting	65 sec	25 sec
Add file to test	196 sec	106 sec
Remove file to test	40 sec	15 sec
Update action recordings	453 sec	418 sec

Table 6.5: Results of maintenance work not based on requirements.

Add a setting: To add a setting to test, to the test artifact based on the old structure took 280 seconds, compared to 166 seconds for the test artifact based on the proposed structure. This means that the time to add a setting to the test artifact based on the proposed structure took 59% of the time to add a settings to the test artifact based on the old structure. Maintenance of the test artifact based on the old structure therefore took 1,69 times more time than of the test artifact based on the proposed structure.
Remove a setting: To remove a setting to test, from the test artifact based on the old structure took 65 seconds, compared to 25 seconds for the test artifact based on the proposed structure. To remove a setting from the test artifact based on the proposed structure therefore took 38% of the time to remove a settings from the test artifact based on the old structure. This means that maintenance of the test artifact based on the old structure took 2,6 times more time than of the test artifact based on the proposed structure.
Add file to test: To add a file to test took 196 seconds for the test artifact based on the old structure, and 106 seconds for the test artifact based on the proposed structure. Therefore, to add a file to test for the test artifact based on the proposed structure took 54% of the time it took to add a file to test for the test artifact based on the old structure. This means that to maintain the test artifact based on the old structure took 1,85 times more time than to maintain the test artifact based on the proposed structure.
Remove file to test: To remove a file to test from the test artifact based on the old structure took 40 seconds, compared to 15 seconds for the test artifact based on the proposed structure. This means that to remove a file to test from the test artifact based on the proposed structure, took 38% of the time to remove a file to test from the test artifact based on the old structure. Therefore, maintenance of the test artifact based on the old structure took 2,67 times more time than of the test artifact based on the proposed structure.
Update action recordings: To update action recordings for the test artifact based on the old structure took 453 seconds, compared to 418 seconds for the proposed structure. Therefore, to update action recordings on the test artifact based on

the proposed structure took 92% of the time to update action recordings on the test artifact based on the old structure. This means that maintenance of the test artifact based on the old structure took 1,08 times more time than of the test artifact based on the proposed structure.
Comparison of total maintenance time between old- and proposed structure: Total time for maintenance work for the test artifact based on the old structure is 1034 seconds, compared to 730 seconds for the test artifact based on the proposed structure. To maintain the test artifact based on the proposed structure therefore took 71% of the time to maintain the test artifact based on the old structure. Therefore, to maintain the test artifact based on the old structure took 1,42 times more time than to maintain the test artifact based on the proposed structure.

6.4.2 Maintenance work based on requirements

This sub-section will present results of the quasi-experiment on maintenance cost, and will focus on maintenance work connected to requirements. First, results of the maintenance work on the test artifact based on the old structure are presented. Secondly are results of the maintenance work on the test artifact based on the proposed structure presented. Lastly is a comparison between the old- and proposed structure given.

6.4.2.1 Old structure

Total maintenance time for each feature: To maintain the test artifact based on the old structure took 1805 seconds for FeR2, which means that the mean time is 226 seconds for each modified requirement of FeR2. FeR1 on the other hand took 762 seconds, which means that the mean time is 95 seconds for each modified requirement of FeR1. Therefore, to maintain the second feature which is FeR1 took 42% of the time to maintain the first feature which is FeR2. To maintain the first feature therefore took 2,37 times more time compared to maintaining the second feature.
Maintenance time for each feature including same FR: When only functional requirements which are the same in both features are included, i.e FR1, FR2, FR4, FR5, and FR7, total maintenance time measured is 1371 seconds for FeR2, which means that the mean is 228,5 seconds for each modified requirement of FeR2. For FeR1 is the total maintenance time 343 seconds, which means that that the mean is 57 seconds for each modified requirement of FeR1. Therefore, to maintain the second feature which is FeR1 took 25% of the time to maintain the first feature which is FeR2. This means that to maintain the first feature took 4 times more time compared to maintaining the second feature.
Maintenance time for each feature excluding same FR: When only functional requirements which are not the same in both features are included, i.e FR3, and FR6, total maintenance time measured is 434 seconds for FeR2, which means that the mean is 217 seconds for each modified requirement of FeR2. For FeR1 is the total maintenance time 419 seconds, which means that that the mean is 209,5 seconds for each modified requirement of FeR1. Therefore, to maintain the first feature which is FeR2 took 97% of the time to maintain the second feature which is FeR1. This

means that to maintain the second feature took 1,03 times more time compared to maintaining the first feature.
Total time for old structure: The total time to maintain the test artifact based on the old structure is 2567 seconds. The mean maintenance time for the test artifact based on the old structure is therefore 256,7 seconds for each modified requirement.

Modified requirement	FeR2	FeR1
FR1	92 sec	30 sec
FR2	325 sec	67 sec
FR3.2, FR3.1	280 sec	242 sec
FR4	138 sec	68 sec
FR5	357 sec	55 sec
FR6.2, FR6.1	154 sec	177 sec
FR7 added	411 sec	87 sec
FR7 removed	48 sec	36 sec

Table 6.6: Results of maintenance work based on requirements on the old structure.

6.4.2.2 Proposed structure

Total maintenance time for each feature: To maintain the test artifact based on the proposed structure took 467 seconds for the first feature which is FeR2, which means that the mean is 58 seconds for each modified requirement of FeR2. For the second feature which is FeR1, took the maintenance 172 seconds, which means that the mean is 22 seconds for each modified requirement of FeR1. To maintain the second feature which is FeR1 therefore took 37% of the time to maintain the first feature which is FeR2. To maintain the first feature therefore took 2,8 times more time than to maintain the second feature.
Maintenance time for each feature including same FR: When only functional requirements which are the same in both features are included, i.e FR1, FR2, FR4, FR5, and FR7, total maintenance time measured is 359 seconds for FeR2, which means that the mean is 60 seconds for each modified requirement of FeR2. For FeR1 is the total maintenance time 51 seconds, which means that that the mean time is 8,5 seconds for each modified requirement of FeR1. Therefore, to maintain the second feature which is FeR1 took 14% of the time to maintain the first feature which is FeR2. This means that to maintain the first feature took 7,04 times more time compared to maintaining the second feature.
Maintenance time for each feature excluding same FR: When only functional requirements which are not the same in both features are included, i.e FR3, and FR6, total maintenance time measured is 108 seconds for FeR2, which means that the mean is 54 seconds for each modified requirement of FeR2. And for FeR1 is the total maintenance time 121 seconds, which means that that the mean time is 60,5 seconds for each modified requirement of FeR1. Therefore, to maintain the first feature which is FeR2 took 89% of the time to maintain the second feature which is FeR1. This means that to maintain the second feature took 1,12 times more time compared to maintaining the first feature.

Total time for proposed structure The total time to maintain the test artifact based on the proposed structure is 639 seconds. The mean maintenance time for the test artifact based on the proposed structure is therefore 63,9 seconds for each modified requirement.

Modified requirement	FeR2	FeR1
FR1	33 sec	0 sec
FR2	30 sec	0 sec
FR3.2, FR3.1	51 sec	72 sec
FR4	32 sec	0 sec
FR5	89 sec	0 sec
FR6.2, FR6.1	57 sec	49 sec
FR7 added	151 sec	29 sec
FR7 removed	24 sec	22 sec

Table 6.7: Results of maintenance work based on requirements on proposed structure.

6.4.2.3 Comparison between old- and proposed structure

Total time to maintain the test artifact based on the proposed structure is 25% of the total time to maintain the test artifact based on the old structure.
Total time to maintain the first feature requirement which is FeR2 on the test artifact based on the proposed structure is 26% of the total time to maintain the same feature requirement on the test artifact based on the old structure.
Total time to maintain the second feature requirement which is FeR1 on the test artifact based on the proposed structure is 23% of the total time to maintain the same feature requirement on the test artifact based on the old structure.

7 Discussion

This chapter will provide a discussion on the results presented in chapter 6, and when applicable tie it to previous research. The chapter is divided according to the four research questions, and will discuss the result of each research question accordingly. Section 7.1 provides a discussion on the first research question on how alignment can be improved and which are the challenges to alignment. Section 7.2 is discussing the second research question on how the proposed structure can support alignment. Section 7.3 focuses on discussing the third research question on how the initial testing cost is affected by the proposed structure. Lastly in section 7.4 is the fourth research questions on how the maintenance cost is affected by the proposed structure discussed.

7.1 RQ-1: How can alignment be improved and which are the challenges?

This section will discuss the results presented in section 6.1 on how alignment can be improved and which are the challenges to alignment. The section is divided so that the results of how alignment can be improved and its benefits are first discussed, and after this are the results of challenges to alignment discussed.

7.1.1 Improve alignment and its benefits

According to the interviewees of the case company are testing and requirements the two most important areas to start focusing alignment improvements on. Testing and requirements are reported by 4 out of 7 interviewees each as prerequisites for improved alignment between requirements engineering and software testing. Suggestions for how to improve alignment for testing includes to actually create tests, and best case scenario is to create automatic tests. Suggestions for requirements are at the same basic level, i.e write down requirements, increase requirements control, and break down requirements. The interviewees pointed out the importance to have the basics in place before other alignment improvements come in question. This is understandable as these two areas are the foundations for further improvements to alignment. Nevertheless are none of the alignment improvements of the two aforementioned areas reported by either Uusitalo et al. (2008), nor Bjarnason et al. (2014), despite their importance. Uusitalo et al. (2008) presents practises used by five different software companies, while Bjarnason et al. (2014) presents practises used by six different software companies, which is why it is remarkable, especially considering that agile software development with reduced documentation etc. becomes more and more popular. Most probably are the two areas already well established at the studied companies, which is why the interviewees do not recognise their importance for alignment, and why they have not been reported in previous research. Despite the two aforementioned areas are alignment improvements suggested by the interviewees of this study very much alike the ones reported by Uusitalo et al. (2008) and Bjarnason et al. (2014). It is important that testers are brought closer to the requirements of the software, either by including them in some parts of the requirements engineering process, or by facilitating communication between areas so that testers are provided with a channel to retrieve information about requirements, which is also reported by Uusitalo et al. (2008). Traceability between requirements and tests is also observed by previous studies, either by tools which is also reported by Bjarnason et al. (2014), or generally between requirements and tests which is also reported by both Uusitalo et al. (2008) and Bjarnason et al. (2014). Traceability can, among other things, increase awareness of requirements coverage making it easier to also increase requirements coverage, as well as facilitate maintenance of software tests when requirements are changed.

7.1.2 Alignment challenges

Resources are limited during software development, which makes prioritisation necessary. Resources can thereby become a challenge to alignment as several different areas want access to them, which is also reported by the interviewees. Interviewees reported that resources are often focused on development, rather than on requirements engineering and software testing, making resources a challenge to alignment. Resources are most probably focused on development as it results in something concrete, visible, and straight forward for customers to estimate development progress on. Requirements engineering and software testing on the other hand help to ensure that the correct product, with high quality is developed. It is therefore a challenge to redirect resources from development, to requirements engineering and software testing, as it reduces the concrete and visible results which are presented to the customers. Bjarnason et al. (2014) also noticed the challenge of resources, where interviewees reported that not enough resources are allocated to testing, considering the large amount of requirements defined on the software. Even if resources are focused on requirements engineering and software testing, is respective area a challenge to alignment. Every interviewee mentioned at least one alignment challenge connected to requirements. The company is practising agile software development, meaning that comprehensive documentation is not created for requirements, which is the main reason why challenges connected to requirements are reported. As requirements are not fully documented, they become undefined or incomplete, hard to break down, and are thereby left open for interpretation. Closely related challenges have been reported in Bjarnason et al. (2014) and Larsson and Borg (2014), namely: defining complete requirements, and defining clear and verifiable requirements. The difference is that the challenges reported by Bjarnason et al. (2014) and Larsson and Borg (2014) are related to defining requirements, while the challenges reported by the interviewees of this research are related to existing requirements.

The third alignment challenge connected to requirements mentioned in Bjarnason et al. (2014) and Larsson and Borg (2014) is about keeping requirements up to date. This challenge is also reported by the interviewees of this study, but is categorised as change management in this research due to that it has to do with handling changes. Verifying the software against its requirements with software testing is also reported by the interviewees as a challenge to alignment. When requirements are not written down, it becomes challenging to verify the software against them, and it is easy that the software instead is verified against requirements which have evolved during software development, meaning that the software is not verified against the correct requirements elicited from stakeholders. Verification of the software could then return a false positive, and the wrong software is delivered to the customers. When requirements are not written down, interactions, communication, and people becomes highly important as a way to share information about requirements between people, and areas. Interactions, communication, and people can on the other hand be challenging to alignment. Misunderstandings can lead to that the wrong information flows between people, while lack of communication lead to a reduced flow of information between people, as well as between areas. Previous research has not mentioned this type of challenge to alignment, yet it is highly relevant, especially with the increase in interactions and communication that comes with agile software development. Traceability is an alignment challenge reported both in this research, as well as in Bjarnason et al. (2014). To create traces between requirements and tests is challenging, especially when requirements are not documented, and when testing is mainly carried out manually, as one of the ends is missing.

7.2 RQ-2: How does the proposed structure support alignment?

A focus group with practitioners was conducted at the company to evaluate how the proposed structure support alignment. This section will discuss the results presented in section 6.2 on how the proposed structure support alignment.

The proposed structure adopt a similar structure as requirements, i.e tests represents features which consist of test modules representing functional requirements. This test structure support alignment in several ways according to practitioners, both alignment practises, as well as challenges to alignment. Interestingly did none of the practitioners report any negative sides with the proposed structure with respect to alignment. The practitioners discussed that developing test artifacts based on the proposed structure requires a bit more time, but they could not think of any negative aspects of the proposed structure with respect to alignment. It is unlikely that this has to do with the design of the research, as the practitioners were given hands on experience of the proposed structure before the focus group was carried out, and were encouraged to discuss both positive and negative sides with the proposed structure during the focus group. Instead, the results are most probably related to that the practitioners do not believe that the proposed structure has any negative impact on alignment. Bjarnason et al. (2014) and Larsson and Borg (2014) reported that to define a good verification process is a challenge to alignment, and similar did interviewees of this research. With the proposed test structure are testers supplied with a clear and structured way for how to create and execute tests to verify the software against its requirements. Tests are divided into smaller pieces with a clear goal, to verify the software against its functional requirements, while at the same time verifying the software against its feature requirements. This reduces the test complexity as it divides the problem into manageable pieces, which can be solved independently to achieve the larger objective. Test creation, as well as software verification against functional and feature requirements are thereby supported by the proposed structure. As the proposed structure is accustomed a similar structure as requirements, are discussions about requirements and tests between areas also facilitated. Creating tests based on the proposed structure does also introduce traces between feature requirements and the tests, as well as between functional requirements and the test modules. A one-to-one mapping is thereby created, increasing awareness of which feature requirements and functional requirements that are covered by the tests. Not only is awareness of requirements coverage increased, it also assists to increase requirements coverage, as with an increased awareness of which requirements are not covered by tests, it is easier to increase requirements coverage. Bjarnason et al. (2014), as well as interviewees of this research reported that traceability between requirements and tests is a challenge to alignment, and is thereby supported by the proposed structure. The proposed structure does also support alignment by improved change management. When requirements are changed, tests verifying those requirements need to be updated accordingly. As the proposed structure adopts modularisation, tests become easier to maintain when requirements are modified. Test modules can be maintained individually, reducing the need to redo entire tests when requirements change. Modularisation also enable test modules to be updated a single time at a single place, and the changes are then propagated to all tests composed of the affected test modules, reducing maintenance effort needed. The situation where tests are not updated when the requirements they are verifying are changed, is a challenge to alignment according to Bjarnason et al. (2014) and Larsson and Borg (2014), as well as by interviewees in this research. This challenge is positively affected by the proposed structure, as the proposed structure ease maintenance, and thereby helps to update tests when requirements change.

7.3 RQ-3: How is the initial development cost affected by the proposed structure?

This section will provide a discussion on the results presented in section 6.3 of how the initial testing cost is affected by the proposed structure. As the results were only analysed using descriptive statistics, it is hard to prove a statistical significance for the findings. Therefore will the discussion be based on indications provided by the results, to discuss how the initial testing cost if affected by the proposed structure.

A pattern is revealed when the development time for the first and second test artifact

are compared to each other with respective test structure. When the first test artifact is based on the proposed structure, and the second test artifact is based on the old structure, is the development time of the second test artifact 21% of the development time of the first test artifact. When the first test artifact instead is based on the old structure, and the second test artifact is based on the proposed structure, is the development time of the second test artifact 89% of the development time of the first test artifact. This indicates that it is easier to create a test artifact based on the old structure when first a test artifact based on the proposed structure has been developed. This could be explained by the fact that as the old structure is not as well structured as the proposed structure, the threshold to learn is therefore higher than for the proposed structure. This is also reflected by a statement from one of the practitioner of the focus group, where it was stated that it is *"easier to create test when the functionality is divided. Will not be as messy and complicated"*. Results of the quasi-experiment therefore indicate that the initial development cost is slightly higher for test artifacts based on the proposed structure, but as it is better structured, the learning threshold is lower than for the old structure. Test artifacts based on the old structure are most probably faster to initially develop because modularisation is not necessary, and therefore can the test artifacts be created without generalisability in mind. Though, this is most probably only true for smaller test artifacts like the ones developed during the quasi-experiment. As the test artifacts grow bigger, modularisation will instead most probably lower the initial development cost. This assumption is not based on any data, rather on sound judgement. One practitioner of the focus group had a similar opinion, stating that: *"When you have created several tests, then you can reuse the parts to save time. It must get better over time, i.e take less time as more parts are created"*
It would therefore be interesting to also let practitioners develop larger test artifacts to compare the difference. Though, as mentioned earlier was it not possible during this research due to lack of resources.
As aforementioned are the results not statistical significant, but they can still provide an indication of how the initial development cost is affected by the proposed structure.

7.4 RQ-4: How is the maintenance cost affected by the proposed structure?

This section will provide a discussion on the results of how maintenance cost is affected by the proposed structure. As presented in section 6.4, two different types of maintenance work were carried out, one based on requirements, and one not based on requirements. Results of both types of maintenance work were only analysed using descriptive statistics, meaning that it is hard to prove a statistical significance for the findings. Nevertheless did the results indicate that the proposed structure positively affects maintenance cost, and this is what will be discussed in this section. Discussion will be divided according to the two types of maintenance work, and then summarised in the end.

7.4.1 Maintenance work not based on requirements

Five different kinds of maintenance work not based on requirements were carried out on the two test artifacts, and as presented in section 6.4.1 is the test artifact based on the proposed structure more efficiently maintained for all five kinds of maintenance work. The total time to maintain the test artifact based on the proposed structure is 71% of the test artifact based on the old structure. The difference in maintenance time between the two structures is most probably due to modularisation used in the proposed structure, where functionality is divided into modules, each defined only at a single place, making it possible to efficiently reuse and modify code. Modularisation is not present in the old structure, resulting in extra work when the same kind of maintenance work has to be carried out at several different places, and does thereby consume more time. If we look closer at the first four kinds on maintenance work, i.e add/remove a setting, and add/ remove a file to test, is the maintenance time for the proposed structure 38%-59% of the maintenance time for the old structure. These have their functionality uniquely defined in modules in the test artifact based on the proposed structure, but not in the test artifact based on the old structure. If we instead look at the last kind of maintenance work which is to update action recordings, is the maintenance effort for the test artifact based on the proposed structure 92% of the maintenance effort for the test artifact based on the old structure. Attributes for action recordings, which are the ones who are updated when maintenance work is carried out on action recordings, are not module specific, and thereby is the maintenance effort similar for both of the two test artifacts.

7.4.2 Maintenance work based on requirements

It is found that the test artifact based on the proposed structure is more efficiently maintained also for maintenance work based on requirements. As presented in section 6.4.2 is the total time to maintain the test artifact based on the proposed structure 25% of the total time to maintain the test artifact based on the old structure. This difference is evenly distributed between the two maintained feature requirements, where the total time to maintain the first feature requirement on the test artifact based on the proposed structure is 26% of the total time to maintain the same feature requirement on the test artifact based on the old structure, and the time to maintain the second feature requirement on the test artifact based on the proposed structure is 23% of the total time to maintain the same feature requirement on the test artifact based on the old structure. The first impression is that the proposed structure is generally more effective to maintain than the old structure, and that subsequent features are not further affected. On the other hand, if we look closer at the modified requirements of each feature, then we get a different picture. Three different groups of functional requirements can be distinguished for the two features: functional requirements which are the same between the two features, functional requirements which are not the same between the two features, and functional requirements which are added/removed between the two features.
Functional requirements which are the same between the two features do only need to be maintained once on the test artifact based on the proposed structure due to modularisation. This does not apply to the test artifact based on the old structure,

where requirements still need to be maintained for the second feature, but as it has already been accomplished for the first feature, is the learning threshold reduced, as well as that code can be reused, which slightly reduces maintenance effort on the second feature.
The second group of functional requirements includes requirements which are not the same between the two features. These need to be maintained for both features, no matter which test structure the test artifact is based on.
The third group of functional requirements includes requirements which are added or removed from the features. For the test artifact based on the old structure this means to carry out the same type of maintenance work for both features, but as knowledge increases between the two features, and as code reuse is possible, the maintenance effort for the second feature is slightly lower than for the first feature. For the test artifact based on the proposed structure on the other hand are two types of maintenance work needed: maintenance of test module, and maintenance of the test including the test module. On the first test are both types of maintenance needed, while on the second test is only maintenance of the test itself needed. This reduces the maintenance time for the second test, but as still some maintenance is needed, the maintenance time is not reduced to zero.
Therefore can we suppose that subsequent maintenance is supported by the proposed structure when features share the same functional requirements (more when functional requirements are modified, and a bit less when they are added or removed).

7.4.3 Summary

Maintenance is supported by the proposed structure, both for maintenance work based on requirements, as well as for maintenance work not based on requirements. Maintenance is supported as the proposed structure adopts modularisation, which divides functionality into modules, making it possible to modify the code at a single place for the first feature, and then automatically propagate the modifications to other features consisting of the functional requirement, reducing the maintenance effort for subsequent features.

8 Conclusion

The objective of this research is to explore different ways to achieve alignment and challenges to alignment, as well as propose and evaluate a test structure connecting functional and feature requirements to support alignment between requirements engineering and automated system tests. A series of semi-structured interviews were conducted with seven practitioners to explore different ways to achieve alignment, as well as challenges to alignment. The proposed structure was evaluated in three different ways: a quasi-experiment with five practitioners on initial development cost, a quasi-experiment on maintenance cost, as well as a focus group with five practitioners to evaluate how the proposed structure support alignment.

The most prominent findings of RQ1 - How can alignment be improved and which are the challenges, are: 1) Alignment can be improved by increasing requirements control, break down requirements into manageable pieces, as well as to write down requirements. This way are testers provided with a clear, structured, and manageable specification of requirements to verify the software against. 2) Alignment can also be improved by the creation of tests to verify the software against its requirements. Automatic tests are preferable as they in a more structured way verifies the software against its requirements, as well as provide rapid feedback of verification. 3) Introducing traces between requirements and tests improves alignment, and can be achieved by creating a specific test for each requirement. 4) Alignment challenges related to requirements are the most experienced, where undefined or incomplete requirements, requirements breakdown, and interpretation of requirements are challenging. 5) Testing is also challenging to alignment, more precisely verification of the correct requirements. 6) To establish traces between requirements and tests is seen as a challenge to alignment. 7) Prioritise requirements engineering and software testing, as well as alignment between the two is challenging as resources are limited.

The most notable results of RQ2 - How does the proposed structure support alignment, are: 1) Change management is supported by the proposed structure due to modularisation, as well as one-to-one mapping between feature requirements and tests, and between functional requirements and test modules. 2) Traceability is supported as when tests are created, traces to functional and feature requirements are introduced. 3) The proposed structure defines a clear and structured way for how to create and execute tests to verify software against its requirements, which increase requirements coverage, ease test creation, as well as supports requirements verification. 4) Organisation and processes will also be supported by the proposed structure as it provides a similar structure for requirements and tests, which facilitates discussions about their connections and dependencies etc.

The most salient findings of RQ3 - How is the initial development cost affected by

the proposed structure, are: 1) The proposed structure reduces the complexity and confusion related to tests, making it easier to initially develop tests for unaccustomed practitioners, but still at a slightly higher cost compared to the old structure. 2) For accustomed practitioners is the initial development cost higher for tests based on the proposed structure.
The findings for RQ4 - How is the maintenance cost affected by the proposed structure, are: 1) Maintenance cost is reduced for test artifacts based on the proposed structure as it adopts modularisation, which makes it possible to modify the code at a single place, and then automatically propagate it to all affected tests.

8.1 Implications for practitioners

Practitioners can use this research as inspiration for how alignment can be improved, as well as which challenges that need to be addressed when the objective is to improve alignment. The proposed structure can be directly adopted by the industry to reduce maintenance cost for tests, while at the same time improve alignment. Improved alignment will in turn coordinate the functioning of requirements engineering and software testing for optimised software development.

8.2 Contributions to academic research

Little research have been carried out in the area of requirements engineering and software testing alignment. This research contributes to academic research by communicating the finding on how alignment between the two areas can be improved, as well as challenges to alignment, and thereby extending the already existing knowledge base regarding requirements engineering and software testing alignment. The research also contributes with an alignment practise which has been evaluated both for efficiency, as well as for how it supports alignment between requirements engineering and automated system tests.

8.3 Future work

This research is focused on connecting functional and feature requirements to support alignment to automated system tests. Quality requirements are excluded from this research as their complexity are not compatible with the time constraints of this research. Nevertheless would it be interesting to evaluate if, and if so in which way quality requirements can be included in the proposed structure to further support alignment to automated system tests. Also, more research need to be carried out to evaluate the efficiency of the proposed structure. This research evaluated the efficiency of the proposed structure by quasi-experiments and using descriptive statistics to provide an indication of its efficiency, nevertheless is it not enough to strengthen the conclusions and generalise the findings, which is left open for future work.

Bibliography

Amannejad, Y. Garousi, V. Irving, R. Sahaf, Z. 2014. A Search-Based Approach for Cost-Effective Software Test Automation Decision Support and an Industrial Case Study. 2014. In *IEEE International Conference on Software Testing, Verification, and Validation Workshops*. p. 302-311. 2014, Cleavland, OH.

Barmi, Z.A. Ebrahimi, A.H. Feldt, R. 2011. Alignment of requirements specification and testing: A systematic mapping study. In *Proceedings 4th International Conference on Software Testing, Verification and Validation Workshops* (ICSTW). IEEE, p. 476–485. 2011, Berlin, Germany.

Berntsson Svensson, R. 2011. Supporting Release Planning of Quality Requirements: The Quality Performance Model. Diss. Department of Computer Science, Lund University.

Bjarnason, E. Runeson, P. Borg, M. Unterkalmsteiner, M. Engström, E. Regnell, B. Sabaliauskaite, G. Loconsole, A. Gorschek, T. Feldt, R. 2014. Challenges and practices in aligning requirements with verification and validation: a case study of six companies. *Empirical Software Engineering* 19 (6): p. 1809-1855. doi: 10.1007/s10664-013-9263-y

Bosch, J. 2000. Design and use of software architectures: adopting and evolving a product-line approach. ACM Press/Addison-Wesley, New York, USA.

Erickson, J. Lyytinen, K. Siau, K. 2005. Agile Modeling, Agile Software Development, and Extreme Programming: The State of Research. *Journal of Database Management* 16 (4): p. 88-100. doi: 10.4018/jdm.2005100105

Hevner, A.R. March, S.T. Park, J. Ram, S. 2004. Design Science in Information Systems Research. *MIS Quarterly*, 28 (1): p. 75-105.

IEEE. 1990. IEEE Standard glossary of software engineering terminology. IEEE Std 610.12-1990, p. 1–84. doi: 10.1109/IEEESTD.1990.101064

IEEE. 2012. IEEE Standard for System and Software Verification and Validation. IEEE Std 1012-2012, p. 1-223. doi: 10.1109/IEEESTD.2012.6204026

Karlsson, L. Dahlstedt, Å.G. Regnell, B. Natt och Dag J, Persson A. 2007. Requirements engineering challenges in market-driven software development - an interview study with practitioners. *Information and Software Technology*, 49(6): p. 588-604. doi: 10.1016/j.infsof.2007.02.008

Kukkanen, J. Väkeväinen, K. Kauppinen, M. Uusitalo, E. 2009. Applying a systematic approach to link requirements and testing: A case study. *In Asia-Pacific Software Engineering Conference. IEEE Computer Society*, Los Alamitos, CA, USA, p. 482–488. doi: 10.1109/APSEC.2009.62

Larsson, J. Borg, M. 2014. Revisiting the challenges in aligning RE and V&V: Experiences from the public sector. In *1st International Workshop on Requirements Engineering and Testing.* Karlskrona Sweden, 26 August. IEEE. p. 4-11. doi: 10.1109/RET.2014.6908671

Manifesto for Agile Software Development. 2015. *Manifesto for Agile Software Development.* [ONLINE] Available at: http://www.agilemanifesto.org/. [Accessed 26 February 2015].

Rani. Misra, R.B. 2004. On determining the software testing cost to assure desired field reliability. *India Annual Conference*, 2004. Proceedings of the IEEE INDICON 2004. p. 517.52. doi: 10.1109/INDICO.2004.1497809

Robertson, S. Robertson, J. 1999. Mastering the Requirements Process. Addison-Wesley, Virginia, ACM Press.

Robson, C. 2002. Real World Research. Second edition, Blackwell publishing.

Royce, W.W. 1970. Managing the development of large software systems: Concepts and techniques. In *Proceedings of the IEEE WESTCON*, p. 1-9

Runeson, P. Höst, M. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering* Springer US 14 (2): p. 131-164. doi: 10.1007/s10664-008-9102-8

Runeson, P. Höst, M. Rainer, A. Regnell, B. (2012) Case study research in software engineering: Guidelines and examples. John Wiley & Sons. Hoboken, New Jersey.

Sabaliauskaitė, G. Loconsole, A. Engström, E. Unterkalmsteiner, M. Regnell, B. Runeson, P. Gorschek, T. Feldt, R. 2010. Challenges in Aligning Requirements Engineering and Verification in a Large-Scale Industrial Context. 16th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2010) In *Requirements Engineering: Foundation for Software Quality /Lecture Notes in Computer Science* 6182. p. 128-142. Essen, Germany

Sommerville, I. 2007. Software Engineering. 8th edi. Addison-Wesley.

Srikanth, H. Cohen, M.B. 2011. Regression testing in software as a service: An industrial case study. *Proceedings of the 2011 27th IEEE International Conference on Software Maintenance* (ICSM '11), IEEE, Williamsburg, VI, 2011; p. 372–381. doi: 10.1109/ICSM.2011.6080804

Unterkalmsteiner, M. Feldt, R. Gorschek, T. 2014. A taxonomy for requirements engineering and software test alignment. *ACM Transactions on Software Engineering and Methodology* 23 (2): p. 1–38. doi: 10.1145/2523088

Uusitalo, E.J. Komssi, M. Kauppinen, M. Davis, A.M. 2008. Linking Requirements and Testing in Practice. *16th IEEE International Requirements Engineering Conference*, p. 265–270. IEEE Computer Society, Barcelona. doi: 10.1109/RE.2008.30

Van Lamsweerde, A. 2001. Goal-oriented requirements engineering: a guided tour. In *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, p. 249-261, Toronto doi: 10.1109/ISRE.2001.948567

Verifying Code by Using UI Automation. 2015. Verifying Code by Using UI Automation. [ONLINE] Available at: https://msdn.microsoft.com/en-us/library/dd286726.aspx. [Accessed 3 February 2015].

Wohlin, C. Runeson, P. Höst, M. Ohlson, M.C. Regnell, B. Wesslén, A. 2000. Experimentation in Software Engineering: An introduction. Kluwer Academic.

Yin, R.K. 2003. Case study research: design and methods. 3rd edi. Sage Publications.

A
Interview questions

Interview Guide

Step 1

Consent information is explained for the interviewee.

- The interview will cover requirements, test, and alignment between them.
- The answers will be treated confidentially.
- Your participation in the study is optional.
- You have the right to whenever you wish, cancel your participation.

Step 2

Introduction: discussion of the interviewee work situation.

1. Kindly tell us about your work
   a. Role
   b. Daily tasks
   c. Responsibility
2. Which roles/other persons do you cooperate the most with in your daily work? How?

Step 3

The interviewer presents the goals of the research.

3. Can you tell us your views of RE and Test alignment?
4. What do you see are the challenges in the alignment; can you perhaps give some examples?

Step 4

The interviewer presents the conceptual model and explain his view of RE and Test alignment.

5. How does the conceptual model map into the context of your company?

Step 5

General questions about requirements and testing.

6. How is your requirements engineering process designed? (Elicitation, specification, prioritisation etc)?
7. In which development phases are the requirements evaluated, and how?
8. When and how is it decided what shoul?

Step 6

Questions on alignment between requirements and testing

9. What is used now to align requirements and testing?
   a. Documents
   b. Processes, methods
   c. Tools
   d. Principles and practices
10. Who is responsible for alignment? (organizational processes, units and roles)?
11. Are testers participating in requirements verification?
12. Is the test strategy based on requirement documents?
13. What is done to maintain alignment between requirements and tests?
14. What are the current problems/challenges in terms of alignment? (What does not work now?)

Step 7

Change related questions.

15. What happens when a requirement is changed (or deleted)?
16. How are the testing artefacts modified when requirements are changed (and vice versa)?

Step 8

Questions related to evaluation

17. How is (good or bad) alignment measured?

Step 9

The interviewer asks about possible solutions and ideas.

18. Do you have any other ideas to propose to improve the alignment? It can be process, methods, tools.
19. What are the expected benefits of an improved alignment?

Step 10

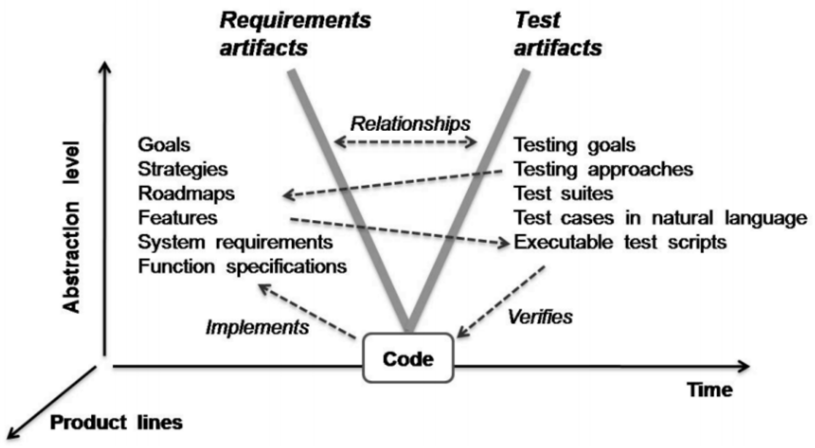
Conclusions.

20. Can you think of any challenges/good things etc. that I have not covered that you think I should have asked?
21. If we are going to continue investigating this area and learning more, are there other roles I should talk to? Some specific names?

Step 11

The interviewer concludes saying that the interview record will be transcribed, summarised and communicated to the interviewee for confirmation and interpretation. The interviewer thanks the interviewee and say good-bye.

V - Model



B
Coding guide

Codes v.0.7

I. Coding instructions

- Code as much of the transcript as possible, even if you think that the text is not very relevant to "alignment"[1]. Since interviews are focussed on alignment, everything said during interview could be useful for further analysis. Code questions as well.
- Codes are prioritized. Please use the priorities in the following way:
  - High level codes – assign the code with highest priority.
  - Medium level codes – choose the code which fits best. Since 2 columns are used for coding ("Primary code" and "Secondary code"), use the "Primary code" column for the code that fits best, and the "Secondary code" column for second best fitting code. If more than 2 codes are applicable – write it in "Comments" field.
- Use "Comments" field as much as possible. It can be used for:
  - describing additional code categories (if code category is not in the list but you think it is important to have it);
  - listing other applicable codes;
  - explanations of "why" you chose particular code;
  - your interpretations of the text;
  - advices you may have for further analysis.

II. Codes

Transcripts should be coded using 3 levels of codes:

1. ***High*** abstraction level codes, based on research questions;
2. ***Medium*** level codes, based on code grouping – code categories;
3. ***Low*** level code is your interpretation of the text (in the Comments field), which is a statement summary.

II.1. High level codes: research questions

Priority range: 1 (highest) – 3 (lowest)
If several research questions are applicable to the same text statement in a transcript, it should be coded by using the code with the highest priority. The other codes should be mentioned in "Comments" column.
Try to use "P" and "B" codes as much as possible.

Priority	Research Question	Description
1	P - Problems, challenges	Problems and challenges related to alignment, or absence of alignment. Also includes "bad" practices.
2	B – experienced and expected Benefits	Suggestions on the way to improve alignment and expected benefits to be achieved by it. Also, current "good" alignment practices and their benefits.
3	C - Current alignment practices	Current practices – the description on how the alignment between requirements and V&V is handled. If a person mentions difficulties or problems related to current practices, then it should be coded as "P – Problems, challenges". If the text is about good current practices and their benefits, it should be coded "B – experienced and expected Benefits".

II.2. Medium level codes: categories

Text should be assigned one or two of the following categories.
Text is coded in 2 columns – "Primary code" and "Secondary code":

- If only one code is applicable – write it down in the "Primary code" column;
- If more than one code is applicable – write the best fitting code into "Primary code" column, and the second best suitable into the "Secondary code" column;
- If more than 2 codes are applicable – write it in "Comments" field.

If the category you want to assign is not among 1-12 categories, please use the 13th category "Other" and describe it in the "Comments" column.

[1] Definition of alignment: Alignment between requirements and V&V is how methods, tools, processes, artefacts, measures, roles, practices, etc., are used in companies in order to coordinate requirements and V&V. It also includes reviews, such as requirements reviews by testers in order to check requirements testability and/or coverage by test cases.

No	Category Name	Description
1	TR - Traceability	Requirements–testing traceability. Includes not only traceability between requirements and test artifacts, but also traceability among requirements artifacts at different abstraction level, and among testing artifacts, as well as the traceability to code or source.
2	IC - Interactions, communication, people	Anything related to people or units interaction. Also includes interactions with vendors, suppliers. Communication barriers and lack of communication should be assigned this category as well.
3	FG - Feedback gathering	Feedback gathering related information, e.g. lessons learnt. Includes information on how feedback from project participants is being gathered and used within organization. This code particularly regards the information on how the interviewees receive feedback about their work. Also includes feedback from post-mortem analysis and from process assessments.
4	OP - Organization, processes	Anything related to organization structure, roles, processes, stakeholders. Also includes information regarding organizational change or process change.
5	CM - Configuration management	Information related to configuration management. Includes change management, change control, etc. Also includes the attitude towards changes. Does not include organization change – organizational or process changes should be assigned “Organization, processes” category.
6	PQ - Product quality aspects	Anything related to product quality. Could include quality requirements, reviews, as well as any quality related issues, such as importance of quality, lack of quality, tradeoffs between quality and functional requirements, system architecture and costs, etc. Process quality should be assigned “Organization, processes” category.
7	ME - Measurements	Any information about measurements related to artifacts, products, processes, etc. Includes key performance indicators, measurement needs, etc.
8	DE - Decisions	Anything related to decision making and distribution. If a particular role is in charge of it, please assign it to the “Organization, processes” category, otherwise use this category. This category is useful for coding information “between chairs” – things that can not be clearly assigned to a role.
9	TL - Tools	Includes information regarding tools, also presence or absence of tools.
10	RQ - Requirements	Anything related to requirements, excluding change management, process, quality requirements and tools, since these are separate categories.
11	TE - Testing	Anything related to testing, excluding change management, process and tools.
12	AR - Artifacts	Information about artifacts – all but requirements and testing. Includes architectural, design artifacts, code, etc.
13	OT - Other	Category that is not among 1-12 categories, but you consider it important. Please describe it in the “Comments” column.

Furthermore, when applicable, text should be assigned one of these categories. Use these codes only in cases when it is clear that information falls into these categories.

No	Extra-Category Name	Description
A	PL - Product lines engineering	Product lines engineering related information - anything regarding variability or product delta or domain (platform, generic components) or application (configurations, customized, unique end products) engineering.
B	OU - Outsourcing	Outsourcing related information.
C	OS - Open source	Open source related information.
D	AG – Agile	Agile software development related information.

III. Coding Example

Text	High and Medium Level Coding				Comments: Low Level Coding
	High Level	Medium Level			
	Research Question (1-3)	Group I (cat. 1-13)		Group 2 (cat. A-D)	
		Primary	Secondary		
A: Yes we tried to have testers in the requirements reviews, so they are there to kind of see, is this requirement testable?	B – experienced and expected Benefits	PQ – product quality aspects			Current alignment practice: Testers participating in requirements review
A. Variability should be more explicit on a detailed requirements level. A lot of times you have to be very explicit about the things which should be able to vary or not. Traditionally, it has been an area of concern as well. But it should be explicit in detailed requirements.	P - Problems, challenges	RQ - Requirements		PL - Product lines engineering	Variability is not explicitly defined