Requirements Engineering that Balances Agility of Teams and System-level Information Needs at Scale

Rashidah Kasauli Namisanvu

Division of Software Engineering
Department of Computer Science & Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden, 2020
Requirements Engineering that Balances Agility of Teams and System-level Information Needs at Scale

Rashidah Kasauli Namisanvu

Copyright ©2020 Rashidah Kasauli Namisanvu
except where otherwise stated.
All rights reserved.

ISBN 978-91-7905-331-4
Doktorsavhandlingar vid Chalmers tekniska högskola, Ny serie nr 4798.
ISSN 0346-718X

Technical Report No 184D
Department of Computer Science & Engineering
Division of Software Engineering
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden

This thesis has been prepared using \LaTeX.
Printed by Chalmers Reproservice,
To my children.
May you be inspired!
Abstract

Context: Motivated by their success in software development, large-scale systems development companies are increasingly adopting agile methods and their practices. Such companies need to accommodate different development cycles of hardware and software and are usually subject to regulation and safety concerns. Also, for such companies, requirements engineering is an essential activity that involves upfront and detailed analysis which can be at odds with agile development methods.

Objective: The overall aim of this thesis is to investigate the challenges and solution candidates of performing effective requirements engineering in an agile environment, based on empirical evidence. Illustrated with studies on safety and system-level information needs, we explore RE challenges and solutions in large-scale agile development, both in general and from the teams’ perspectives.

Method: To meet our aim, we performed a secondary study and a series of empirical studies based on case studies. We collected qualitative data using interviews, focus groups and workshops to derive challenges and potential solutions from industry.

Findings: Our findings show that there are numerous challenges of conducting requirements engineering in agile development especially where systems development is concerned. The challenges discovered sprout from an integration problem of working with agile methods while relying on established plan-driven processes for the overall system. We highlight the communication challenge of crossing the boundary of agile methods and system-level (or plan-driven) development, which also proves the coexistence of both methods.

Conclusions: Our results highlight the painful areas of requirements engineering in agile development and propose solutions that can be explored further. This thesis contributes to future research, by establishing a holistic map of challenges and candidate solutions that can be further developed to make RE more efficient within agile environments.

Keywords
Requirements Engineering, Systems Development, Coordination, Large-scale Agile, Plan-driven, Empirical Research, Methods Co-existence, Safety-critical System Development
Acknowledgment

Working between two separate environments, Uganda and Sweden, the last four years have been quite an experience for me. I have interacted with many interesting, supportive and knowledgeable people that I would like to convey my gratitude to.

My heartfelt gratitude goes to my main supervisor Eric Knauss for the exceptional supervision that has changed my life in ways I had never envisioned. Through your positive attitude, rigor and confidence in my abilities, I have experienced the true spirit of research. Thank you!

To my co-supervisor Benjamin Kanagwa, I appreciate the encouragement, and calming sentiments that kept me strong. My examiner Ivica Crnkovic, thank you for guiding my research in a great and supportive way.

Special gratitude goes to all my 16 co-authors including Grischa Liebel, Jennifer Horkoff, Francisco Gomes, Rebekka Wohlrab, Jan-Philip Steghöfer, Salome Maro, Agneta Nilsson, Gul Calikli. It has been loads of fun working among you. Thank you for always sharing insightful thoughts and giving constructive feedback whenever asked.

Michel Chaudron, Engineer Bainomugisha and the SIDA BRIGHT project team in Uganda and Sweden. Thank you for making my PhD dream a reality. Special gratitude goes to Pär Meiling for always ensuring we have housing in Sweden. Thank you Pär.

To all colleagues and administrative staff at the software engineering division of Chalmers CSE department, thank you for the friendly environment. Léuson Mario Pedro da Silva, thank you for the relaxing jokes. Alessia Knauss, thank you for helping me stay healthy and sane.

My fellow PhD students on BRIGHT especially Hawa Nyende, Micheal Kizito, Dragule Swaib, David Bamutura and Grace Kobusinge. I have enjoyed the shared frustrations :) and happiness in travelling together.

Colleagues at Makerere SCIT especially Joseph Balikuddembe, Mary Nsabagwa, Jacob Katende, Hasifa Namatovu, Moses Ntanda, Joyce Nakatumba. Thank you for your motivation and support.

My father Hajji M.Z. Kasauli and mothers Cornety Kivumbi, and Rahma Makumbi. Thank you for your constant prayers. I know you are my miracle! My brothers Yasin, Shakie, Umar, Zed, and sisters Madiina, Aisha, Sharifah, Shamirah, Rukia, and Shakirah. The Kasaulis. The onus is now on you. :D

To my husband Bashir Mwebe, thank you for encouraging and supporting me to pursue this degree. You showed me that I could endure and never allowed me to give up. To my children: Nushin, Rahma, Rabiib, Rithwana, and Raihan, thank you for your prayers and endurance.
Alhamdulillah! Alhamdulillah! Alhamdulillah! None of this would be possible without your might Allah. Thank you Allah for putting me in the company of those wonderful and supportive people these last four years. May you continue to bless them all!

**Funding:** This research was funded in part by the Swedish International Development Cooperation Agency (Sida) and Makerere University under Sida contribution No: 51180060; Project No. 317; Building Research Capacity in Innovative Information and Communication Technologies for Development (ICT4D) for Sustainable Socio-economic Growth in Uganda (BRIGHT).
List of Publications

Appended publications

This thesis is based on the following publications:


**Other publications**

The following publications were published during my PhD studies. However, they are not appended to this thesis, due to contents not related to the thesis.


Research Contribution

All the included papers were published with collaborations from colleagues. I am the main author of five of the six included papers and as such responsible for the research design, dividing the work between co-authors and performing most of the writing. In particular, I have participated in the included papers as follows:

In papers A–C, the planning, design, execution of the research and writing were mostly done by me. For paper D and E, I participated in part of the data collection, participated and coordinated the writing of the papers. Paper F is a tool paper which I extend with a technical report expanding on the 2 paged document that was published. Here I, therefore, improve my contribution to include formal documentation of the tool.
# Contents

**Abstract** \( v \)

**Acknowledgement** \( vii \)

**List of Publications** \( ix \)

**Personal Contribution** \( xi \)

## 1 Introduction 1

### 1.1 Background and Related Work 2

#### 1.1.1 RE in Systems’ Software Development 3
#### 1.1.2 Traditional Waterfall Process Vs. Agile Development in Systems Engineering 4
#### 1.1.3 Large-scale Agile Systems Development 7
#### 1.1.4 RE in Large-scale Agile Environments 8
#### 1.1.5 System Understanding in the Large 9
#### 1.1.6 Cross-cutting concerns in Development 10

### 1.2 Research Focus 11

#### 1.2.1 Gain insight in current state of RE 12
#### 1.2.2 Exploring solution space 12

### 1.3 Research Methodology 12

#### 1.3.1 Research Approach 13
#### 1.3.2 Case Study Method 14
#### 1.3.3 Secondary Study 17
#### 1.3.4 Threats to validity 17

#### 1.3.4.1 Construct validity 17
#### 1.3.4.2 Internal validity 18
#### 1.3.4.3 External validity 18
#### 1.3.4.4 Reliability 19

### 1.4 Research Synthesis 19

#### 1.4.1 Paper A: RE Challenges in Large-Scale Agile 19
#### 1.4.2 Paper B: Safety-Critical Systems in Agile 21
#### 1.4.3 Paper C: Agile Islands in a Waterfall 22
#### 1.4.4 Paper D: RE practices in large-scale Agile 23
#### 1.4.5 Paper E: Charting coordination needs 26
#### 1.4.6 Paper F: Tool support for managing requirements 27

### 1.5 Summary of Results 28

#### 1.5.1 Current status of RE (G1) 28
## 1.5.2 Exploring solution space (G2)

### 1.6 Discussion

### 1.7 Conclusions and Future Work

### 2 Paper A

#### 2.1 Introduction

#### 2.2 Background and Related Work

#### 2.3 Research Methodology

#### 2.4 Findings

#### 2.4.1 What are possible scopes of applying agile methods in large-scale system development? (RQ 1)

#### 2.4.1.1 Context of Case Companies

#### 2.4.1.2 Agile Scope in Large-Scale System Development

#### 2.4.2 How is the role of requirements characterized in large-scale agile system development? (RQ 2)

#### 2.4.3 Which requirements related challenges exist in large-scale agile system development? (RQ 3)

#### 2.4.3.1 Shared Understanding of Value

#### 2.4.3.2 Build and Maintain System Understanding

#### 2.5 Discussion and Implications

#### 2.6 Conclusion and Outlook

### 3 Paper B

#### 3.1 Introduction

#### 3.2 Methodology

#### 3.2.1 Search strategy

#### 3.2.2 Inclusion and Exclusion criteria

#### 3.2.3 Data extraction and Synthesis

#### 3.2.4 Limitations and Threats to Validity

#### 3.3 Findings

#### 3.3.1 RQ1: Existing research about agile development of SCS

#### 3.3.2 RQ2: Key benefits of applying agile methods to SCS

#### 3.3.3 RQ3: Challenges with agile development of SCS

#### 3.3.4 RQ4: Solution candidates (e.g. principles and practices) for challenges with respect to agile development of SCS

#### 3.3.5 Synthesis of Findings

#### 3.4 Discussion and Conclusion

### 4 Paper C

#### 4.1 Introduction

#### 4.2 Related Work

#### 4.3 Research Method

#### 4.3.1 Data Collection

#### 4.3.2 Data Analysis

#### 4.3.3 Threats to Validity

#### 4.4 The Case Study

#### 4.4.1 Roles in the Departments

#### 4.4.2 Requirements Model in the Departments

#### 4.5 Challenges and Strategies
## 5 Paper D

### 5.1 Introduction

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>88</td>
</tr>
</tbody>
</table>

### 5.2 Background and Related Work

#### 5.2.1 Large-Scale Agile

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>89</td>
</tr>
</tbody>
</table>

#### 5.2.2 RE and Agile

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>90</td>
</tr>
</tbody>
</table>

### 5.3 Research Methodology

#### 5.3.1 Case Companies

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>91</td>
</tr>
</tbody>
</table>

#### 5.3.2 Sampling and Data Collection

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>93</td>
</tr>
</tbody>
</table>

#### 5.3.3 Data Analysis

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
</tr>
</tbody>
</table>

#### 5.3.4 Threats to Validity

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>97</td>
</tr>
</tbody>
</table>

### 5.4 Pervasiveness of Agile development (RQ1)

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>98</td>
</tr>
</tbody>
</table>

### 5.5 Challenges and Potential solutions (RQ2 and RQ3)

#### 5.5.1 Build and Maintain Shared Understanding of Customer Value

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
</tr>
</tbody>
</table>

##### C1.a: Bridge gap to customer

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
</tr>
</tbody>
</table>

##### C1.b: Building long-lasting customer knowledge

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
</tr>
</tbody>
</table>

#### 5.5.2 Support Change and Evolution

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
</tr>
</tbody>
</table>

##### C2.a: Managing experimental requirements

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
</tr>
</tbody>
</table>

##### C2.b: Synchronization of development

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>105</td>
</tr>
</tbody>
</table>

##### C2.c: Avoid re-specifying, encourage re-use

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>106</td>
</tr>
</tbody>
</table>

##### C2.d: Updating requirements

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>107</td>
</tr>
</tbody>
</table>

#### 5.5.3 Build and Maintain Shared Understanding about System

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>108</td>
</tr>
</tbody>
</table>

##### C3.a: Documentation to complement tests and stories

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>108</td>
</tr>
</tbody>
</table>

##### C3.b: System vs component thinking

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
</tr>
</tbody>
</table>

##### C3.c: Creating and Maintaining Traces

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
</tr>
</tbody>
</table>

##### C3.d: Learning and long-term knowledge

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>112</td>
</tr>
</tbody>
</table>

##### C3.e: Backward compatibility

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>113</td>
</tr>
</tbody>
</table>

#### 5.5.4 Representation of Requirements Knowledge

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>113</td>
</tr>
</tbody>
</table>

##### C4.a: Manage levels vs. decomposition

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>114</td>
</tr>
</tbody>
</table>

##### C4.b: Quality requirements as thresholds

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>115</td>
</tr>
</tbody>
</table>

##### C4.c: Tooling not fit for purpose

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>116</td>
</tr>
</tbody>
</table>

##### C4.d: Accommodate different representations

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>117</td>
</tr>
</tbody>
</table>

##### C4.e: Consistent requirements quality

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>117</td>
</tr>
</tbody>
</table>

#### 5.5.5 Process Aspects

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>118</td>
</tr>
</tbody>
</table>

##### C5.a: Prioritization of distributed functionality

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>118</td>
</tr>
</tbody>
</table>

##### C5.b: Manage completeness

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>119</td>
</tr>
</tbody>
</table>

##### C5.c: Consistent requirements processes

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
</tr>
</tbody>
</table>

##### C5.d: Quality vs time-to-market

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
</tr>
</tbody>
</table>

#### 5.5.6 Organizational Aspects

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>122</td>
</tr>
</tbody>
</table>

##### C6.a: Bridge Plan-Driven and Agile

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>122</td>
</tr>
</tbody>
</table>

##### C6.b: Plan V & V based on requirements

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
</tr>
</tbody>
</table>

##### C6.c: Time for invention and planning

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>124</td>
</tr>
</tbody>
</table>
7.9 Limitations and future work . . . . . . . . . . . . . . 169

Bibliography 171
Chapter 1

Introduction

“It isn’t just that businesses use more software, but that, increasingly, a business
is defined in software. That is, the core processes a business executes—from how
it produces a product, to how it interacts with customers, to how it delivers
services—are increasingly specified, monitored, and executed in software. This
is . . . a transition that is spreading to all kinds of companies, regardless of the
product or service they provide.” – Jay Kreps, CEO of Confluent

Software has pervaded our lives and is continuously gaining importance. Seen
as a driver for innovation, even the formally hardware-based systems
like automobiles are becoming more software-oriented than before [1,2]. This
increased use of software has led to more software-intensive systems, i.e. systems
that consist of software, hardware and possibly mechatronic parts defining the
context in which they are used. Such systems include, e.g., telecommunications
and automotive systems. At the large-scale, for such software-intensive systems
requirements engineering is the key to success [3,4].

Requirements Engineering (RE) is traditionally a sequential process where
the execution of, for instance, software development requires indisputable
completion of the requirements specification phase [3,5]. This traditional
approach to RE has formed the foundation on which many large-scale systems
companies are built. These companies often have to deal with standards and
regulations [6], along with parallel development of hardware and software. With
advancement in software, and new players coming into the market, competition
has increased and customer demands are evolving much faster, making reliance
on traditional (plan-driven) methods, with their long lead times and lack of
flexibility, less of an option. Thus large-scale systems development companies
are seeking better approaches that allow flexibility, a characteristic of agile
development methods.

Although initially meant for development at a small scale [7], agile devel-
opment methods are increasingly adopted by large-scale systems development
companies [8,10]. On top of the flexibility that agile methods provide, their
adoption at scale is driven by reported success in handling changing customer
However, their adoption at scale is challenging, not only because of the scale
but also the foundation on which many of these companies are built that calls
for a sequential adoption [12,14].
Agile development promotes customer collaboration which is in line with RE. Thus, RE and Agile development seem to support each other. However, long upfront analysis—a phase in RE commonly leading to extensive documentation—is considered anti-agile creating some friction between RE and agile methods. Existing work on this friction has addressed practices and their challenges \cite{15,16} while also commenting on synergies and conflicts of traditional RE with agile \cite{17}, without focusing on RE in large-scale agile systems development. This research attempts to address that friction while focusing on large systems’ development.

We approach the problem through a series of empirical studies that discover the information needs and related knowledge, pertinent to systems development. The overall aim of this thesis is to investigate the challenges and solution candidates of performing effective RE in an agile environment, based on empirical evidence. We explore RE challenges and solutions in large-scale agile systems development, both in the general and the teams’ perspectives. The new knowledge and methods presented in this thesis can be used to inform process and tool design in large-scale agile system development. Once the gap between agile and traditional methods is addressed, many challenges relating to coordination and knowledge management will have been combated. Ultimately, large-scale companies have met their agile adoption goals.

The thesis is composed of two parts, the introduction part (Chapter 1) and the second part is an attachment of the included papers. The rest of this chapter is structured as follows: Section 1.1 presents the background and related work of the research presented in this thesis. Section 1.2 presents our research questions. The research methodology is described in Section 1.3 while Section 1.4 provides a synthesis of our research outputs. In Section 1.5, we give conclusions and future work. For the second part, we have Chapter 2 to Chapter 7 with Papers A–F respectively.

1.1 Background and Related Work

The adoption of agile methods has changed the way RE is interpreted in development. According to Leffingwell, “No matter the specific method, agile’s treatment of requirements is fundamentally different” \cite{18}. Whereas some argue that RE can be viewed from two different angles; 1) as a formal and structured transformation of information \cite{17} (e.g. traditionalists) or 2) as a collaborative effort relying on the creativity and competence of the involved engineers \cite{19} (e.g. the agilists), others seem to imply that RE ceased to exist with the introduction of agile methods (e.g. ‘architecturalists’). The dispute on agile methods and RE existence is not something to argue for. In fact, as noted by Paetsch et al. \cite{17}, the RE process phases of elicitation, analysis, and validation are present in all agile processes. Thus in this thesis, we take the view that RE is a collaborative effort of which agile methods are an example.

This section discusses the background of RE and agile development in the systems development context. We start by detailing the RE process in systems development in which we describe the traditional RE process and fundamental RE terminology. We then review the literature on traditional methods and agile software development while discussing the documented comparison of the
two methods. A discussion on large-scale agile systems development comes next followed by a discussion on RE in large-scale agile development. Since RE is a communication problem, we discuss the knowledge management literature before concluding the section with a discussion on safety as a cross-cutting concern in development. Safety systems’ development is dependent on effective RE. Safety is used as a maximum formality example for performing RE in an agile development environment.

1.1.1 RE in Systems’ Software Development

Systems development or engineering was initially about configuring hardware components into physical systems like ships or railroads [20]. The component parts would be produced once the configuration and the requirements specification are done. The production was thus a sequential process. As technology advanced and software began to appear in such systems, the same sequential process of development was naturally followed [20]. Over the years, even as software in systems (e.g., automotive [1, 21]) increased, such sequential processes formed the basis for development. The dependence on software has led to software-intensive systems—systems that depend on software, hardware and the context in which they are operating for correct operations. It is important to note that for such software-intensive systems, software failures are commonly associated with RE challenges [4].

RE is defined as “a systematic and disciplined approach to the specification and management of requirements with the goal to:

[a] Know the relevant requirements, achieve a consensus among the stakeholders about these requirements, document them according to given standards, and manage them systematically

[b] Understand and document stakeholders’ desires and needs

[c] Specify and manage requirements to minimize the risk of delivering a system that does not meet the stakeholders’ desires and needs

All of which address important facets of RE: (1) process orientation, (2) stakeholder focus, and (3) importance of risk and value considerations [22].”

RE activities typically include elicitation, analysis, specification, validation and management, with requirements prioritisation coming in to support elicitation and analysis by identifying the most valuable requirements [23].

Requirements elicitation process includes users getting involved in gathering requirements [15]. During elicitation, the initial information regarding requirements and context is gathered. The requirements analysis phase then follows to check for consistency, completeness, necessity and feasibility of the requirements, thus creating an understanding of the requirements. The next activity is the requirements specification where the requirements are defined in terms of system behaviour, decomposing the problem into component parts and serving as input to design specification. The end of this process is marked with a requirements specification document where the agreed requirements are documented for communication with stakeholders and developers. The requirements validation is important for confirming customers’ needs and correcting errors in the specifications to avoid rework which could be expensive. Requirements
management aims to keep the requirements’ quality whenever there are updates or newly added requirements during the system implementation. The updating of requirements also means dependencies and relationships of different requirements documents must be managed, and all the requirements should also be traceable, which helps to investigate the impact of the changes [5]. The flow of the processes is as represented in Figure 1.1. The requirements flow through a sequential process similar to that followed in the waterfall or V-model methods that have existed in system engineering for many generations.

With increased emphasis on user value as well as increased pace of change, more strain has been put on the traditional, sequential approach. The strain came from the realization that requirements were more emergent with use than pre-specifiable, and thus traditional methods were not suitable for producing user valued products [20]. Alternative methods, like agile methods, were then devised and adopted.

1.1.2 Traditional Waterfall Process Vs. Agile Development in Systems Engineering

The traditional waterfall process is a classical systems engineering process that follows a sequential flow of activities, as shown in Figure 1.2. The phases are cascaded one after another with former processes being frozen when work is continued to later stages. As can be seen in Figure 1.2 once inconsistencies are noticed at the testing phase, then software requirements are revisited, which has proved expensive in the fast-paced change of today. As demonstrated in Figure 1.1 RE also follows a sequential process similar to the waterfall methods.

On the contrary, agile development is both iterative and continuous. According to the agile alliance [25],

“Agile is the ability to create and respond to change in order to succeed in an uncertain and turbulent environment.”
Concerning agile software development, Beck [26] notes that:

"Agile software development is an umbrella term for a set of methods and practices based on the values and principles expressed in the Agile Manifesto [26]."

The agile manifesto identifies four values for agile development as shown in Table 1.1

| 1. | Individuals and interactions | over | processes and tools. |
| 2. | Working software | over | comprehensive documentation. |
| 3. | Customer collaboration | over | contract negotiation. |
| 4. | Responding to change | over | following a plan. |

While the agile advocates acknowledged the items on the right as having value, they valued the items on the left more. The agile manifesto also connects 12 principles that attempt to make the agile values more concrete and deliver solid guidance for software development teams and their projects. The original agile principles are as presented in Table 1.2. These have been reviewed by William [27] but the basic concepts remain the same.

Agile methods like Scrum [28] and XP [29] are based on the above values and principles and encourage flexible, light-weight software development with short iterations [30] thus creating the ability to deal with changing requirements and fast time-to-market. In agile software development, requirements are allowed to evolve through collaboration between self-organizing, cross-functional teams utilizing the appropriate practices for their context.

In agile development, instead of fixing all plans at the project start, the project is broken into smaller sub-tasks which are implemented in short time-boxed iterations [18], commonly referred to as sprints. Sprint duration is also an agile variable with differing recommendations from the different agile methods but typically spans 2–4 weeks. The goal of each sprint is to produce shippable code incrementally. Sprints have the same pattern, which has three common phases illustrated in Figure 1.3
<table>
<thead>
<tr>
<th>No.</th>
<th>Principle</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.</td>
</tr>
<tr>
<td>2.</td>
<td>Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.</td>
</tr>
<tr>
<td>3.</td>
<td>Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.</td>
</tr>
<tr>
<td>4.</td>
<td>Business people and developers must work together daily throughout the project.</td>
</tr>
<tr>
<td>5.</td>
<td>Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.</td>
</tr>
<tr>
<td>6.</td>
<td>The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.</td>
</tr>
<tr>
<td>7.</td>
<td>Working software is the primary measure of progress.</td>
</tr>
<tr>
<td>8.</td>
<td>Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.</td>
</tr>
<tr>
<td>9.</td>
<td>Continuous attention to technical excellence and good design enhances agility.</td>
</tr>
<tr>
<td>10.</td>
<td>Simplicity – the art of maximizing the amount of work not done – is essential.</td>
</tr>
<tr>
<td>11.</td>
<td>The best architectures, requirements, and designs emerge from self-organizing teams.</td>
</tr>
<tr>
<td>12.</td>
<td>At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.</td>
</tr>
</tbody>
</table>

The first phase is a planning phase that includes a review of the sprint backlog that is later (re-)prioritized and estimation of the work to be done established [31]. The team—usually 5-9 people—then commits to the work. The second phase is the development phase in which the team takes responsibility for the requirements and elaborates them. Then the code implementation, building, and testing are done. The last phase is the delivery of the increment and assessment of the sprint [31], sprint review in Fig. 1.3.

For effective development in agile methods, communication, and collaboration among the team members are crucial. Thus the practice of face-to-face communication with team members is recommended [15], together with an on-site customer so that the developers will get quick clarifications on requirements. Most agile methods, such as scrum, have the practice of a daily stand-up meeting for the team. At this meeting, the team members give a report of what has been done, and what they plan to do, including the challenges that they could be facing. In this way, the sprint progress is tracked, and the problems are reported in time.

De Lucia and Qusef [32] state that the main difference between traditional and agile development is not whether, but when to do RE. In the traditional approach, RE is done only at the beginning of development (see Figure 1.2).
while in agile development, RE is as continuous and incremental as the product being developed. Research on agile and traditional methods has varied considerably, starting from comparing characteristics of agile with those of traditional methods \cite{33,35} to uncovering challenges of hybrid development in organisations \cite{36}.

The hybrid methods come from the fact that many large organisations are only still transitioning to agile and thus have both traditional and agile methods in operation. Also, many have safety concerns that call for a well-streamlined process and documentation that would otherwise be ignored in a fully agile process. Theocharis et al. \cite{37} conducted a study to find out whether agility accelerated the extinction of traditional methods. They, however, found a mixed application of methods and concluded that hybrid approaches formed the standard of today’s development. Research in this field has advanced to the point that researchers study how the methods have been combined in development \cite{38}, the challenges of having such combinations \cite{39} and potential solutions to such challenges \cite{40}. It should be noted that in large-scale organisations in practice, adoptions start with the software development teams using agile methods while the rest of the organisation works with traditional methods \cite{41}. Whereas these studies are partially empirical, they lack the RE perspective, which gives a more general focus on communication and coordination in such environments.

1.1.3 Large-scale Agile Systems Development

Although initially meant for small collocated teams, the reported success of agile methods has led to their adoption at scale \cite{11,8,42} and in systems development \cite{9,10,43}. Large-scale companies are characterised by long lead times, many stakeholders with varying backgrounds and needs during (and probably after) the systems’ development. Also being in the context of systems development, large-scale companies are also characterised by stable sequential processes \cite{44}. Because of these characteristics, agile adoption at scale was received with skepticism, with most adoptions starting in software development
teams. This selective adoption created pockets of agile teams within a larger ecosystem of plan-driven culture. We term these pockets of agile teams as *agile islands* (Paper A and C).

Large-scale agile development is a complex term that has been interpreted in various ways \[6\]. It has been used to refer to varying contexts starting from one large team in a project to large multi-team projects \[45\]. Dingsoyr et al. \[46\] provide a taxonomy of scale and categorise large-scale as a company or project with 2–9 development teams. We use the term “large-scale agile system development” to refer to large-scale agile development in the context of systems engineering and define it as follows:

*Large-scale agile system development is the development of a product consisting of software, hardware, and potentially mechatronic components that include more than 6 development teams and is aligned with agile principles.*

Large-scale agile development has received considerable attention from the research community with many reporting successful adoption \[9, 13, 42, 47\] although challenges remain. Several challenges relating to e.g. coordination in a multiple team environment with hierarchical management and organisational boundaries \[8\] and coordinating work between agile teams \[43\] have been identified. As such, large-scale agile frameworks are being adopted to overcome the challenges \[48\].

SAFe framework has been reported by the state of agile report \[49\] as the most popular with 30% of the companies in the survey using it. These frameworks have received considerable attention from the research community recently, with some studies exploring how these frameworks have been adopted (e.g. SAFe \[50\]) and recommended guidance for clear adoption \[51\]. Others have explored the benefits and challenges of adopting these frameworks \[52, 53\]. It, however, remains unclear whether these frameworks do address the challenges identified by the introduction of agile methods at scale.

### 1.1.4 RE in Large-scale Agile Environments

“No matter the specific method, agile’s treatment of requirements is fundamentally different \[18\].”

Since many systems development companies are adopting agile methods, the existing techniques are proving inadequate. As a result, RE becomes an even more significant challenge for agile development companies, especially at scale and in systems development. The research on RE in agile development environments has thus attracted much attention from both research and practice, leading to a need for more empirical studies that devise working solutions collectively.

The process of doing RE in an agile development environment has been coined Agile RE, although it has no unanimously accepted definition \[16\]. Agile RE can however be weakly defined as the agile way of performing RE. Some of the existing research on agile RE has explored the benefits and challenges \[16, 23, 54\], together with the practices used in development \[15, 19\]. The challenges identified include, e.g. neglect of non-functional requirements, customer availability and minimal documentation. The consensus in all the
1.1. BACKGROUND AND RELATED WORK

studies is that agile RE addresses some of the classical RE challenges, e.g. communication gaps, while it also causes new challenges which call for new techniques.

Identified practices that apply to agile RE include: face-to-face communication, customer involvement, requirements prioritization, review meetings and retrospectives, iterative/incremental development, user stories, test-driven development, acceptance tests, change management and code refactoring. The practices adopted by teams vary depending on the agile method of development that is chosen. For instance, in XP, the planning game is used and begins with an on-site customer who writes the requirements. These are later prioritized by the development team together with the customer. However, experience with practitioners has shown that practices and methods have been adopted randomly according to the development needs.

Other researchers have compared the use of traditional RE and agile RE while also identifying how agile development can benefit from traditional methods. Paetsch note that the major difference is in the amount of documentation carried out in the development process. In summary, there is substantial amount of existing work on the use of agile RE and its practices. However, we notice a lack of empirical evidence in terms of large-scale agile systems development.

1.1.5 System Understanding in the Large

Software engineering is a knowledge-intensive endeavour with activities from requirements elicitation to the project coordination and management. It is unlikely for the team members to have all the knowledge obtained from those activities. Agile development adds to the challenge with the idea of breaking down the system requirements into features to be developed every sprint, thus making the bigger picture of system understanding unclear. However, in order to deliver a quality product, the development team has to have a clear understanding of the system. Effective communication and knowledge management become an essential part of their development.

Knowledge management is crucial for proper system understanding and ensuring effective communication helps to transfer knowledge. However, few existing works explicitly address communication in agile development. Communication in agile projects has mainly focused on the impact of the agile practices on communication. In a systematic review, Hummel et al. found reports that agile methods lead to improved communications in large-scale development projects. They also found that the informal communication on which agile methods depend may be problematic for large projects with many stakeholders and a lot of shared information. The studies present a broad understanding of the communication concept without focusing on the social interaction and behavior of teams. Studies suggest synchronous and asynchronous communication means e.g. wikis and group e-mails in order to establish the multiplicity of social links between team members and to provide continual access to project information in large-scale settings. While investigating inter-team coordination mechanisms, Nyrud and Stray find ad-hoc conversations more beneficial than daily stand-up meetings that agile methods recommend. These were found to be the most time-consuming and
involved less coordination. Such coordination can be tagged in artefacts which also could come from both agile and traditional methods [61].

For large-scale development, such artefacts are shared within the team and also across different teams. Some high-level artefacts, e.g. architectural models [62], and tools [63] are used as boundary objects. Boundary objects are defined by Star and Griesemer as follows:

**Boundary objects are objects which are both plastic enough to adapt to local needs and the constraints of the several parties employing them, yet robust enough to maintain a common identity across sites [64].**

Boundary objects are shared between several teams and each team can access what they need from it and thus helping in knowledge management across the system development. As Rolland et al. [6] state “We believe there is a need to emphasize the boundary work and boundary infrastructures that are required for working across contexts resolving and coordinating complex socio-technical interdependencies.” This thesis explores the use of boundary objects in development.

Tools are essential to ensure effective communication and collaboration in large-scale development companies. In this thesis, the focus is on tools that help in effective communication of software requirements. Through a survey subjected to requirements tool vendors, de Gea et al. [65] provide an insight into the degree of support offered by requirements tools and the capabilities of these tools in supporting the RE process. They find a substantial number of tools supporting requirements elicitation but a poor representation of tools for requirements management. de Gea et al. argue that RE tools are traditionally oriented towards textual requirements and thus the reason for fewer modeling tools. These textual requirements tools are, however, used for elicitation processes. This thesis also explores the use of a text-based tool for requirements management.

### 1.1.6 Cross-cutting concerns in Development

We use the term cross-cutting concerns to mean the development concerns that are driven by RE and require system understanding. These are usually quality concerns or non-functional requirements of any software development. Existing research conducted in agile software development identifies challenges of dealing with non-functional requirements [66]. For the context of this thesis, the cross-cutting concern that raises interest is safety since it gives us a maximum formality example.

Safety raises concern since agile methods tend to have less favour for documentation and processes [67,68] yet safety requires a well-defined process with extra documentation for certification [69]. Glinz [22] defines safety as the capability of a system to operate without resulting in harming people, property, or the environment [22].

Safety systems or Safety-critical systems (SCS) are becoming more prevalent in use with the advance of the digital era [70]. Many software applications are highly critical for safety, and are found in, for example, the avionics, medical, railway, and automotive sectors. Examples of such systems include flight
control systems in avionics [71], automatic braking systems in automotive [72], Train Control Management System (TCMS) for a high-speed train in railway systems [73].

Requirements for the development of SCS (Safety requirements) are commonly stated as quality requirements and in some cases also stated in terms of functional requirements and thus usually follow the same development path as all other requirements. However, since these safety requirements have more stringent rules on the testing and validation, extra checks are put in place. With this extra effort demonstrated in safety development, using safety as a comparison gives us a maximum formality example to ensure we do not miss critical elements of RE in system development.

1.2 Research Focus

The overall goal of this thesis is to investigate the challenges and solution candidates of performing effective RE in an agile environment, based on empirical evidence. Considering that the use of agile methods suffers most at scale, we focus on large-scale systems development companies to achieve our goal.

The main goal is broken down to two sub-goals as follows:

- **G.1:** To gain insight in the current state of RE in large-scale agile systems development.

- **G.2:** To recommend solutions for addressing critical cross-cutting challenges of RE in large-scale agile system development.

To meet the set goals while scoping the thesis, research objectives (RO) were defined for each of the goals and addressed as presented in Table 1.3. The connection between the goals and objectives is further illustrated in Figure 1.4.

<table>
<thead>
<tr>
<th>Table 1.3: Research questions for the respective aims</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>G.1: Gain insight in current state of Requirements Engineering</strong></td>
</tr>
<tr>
<td>RO.1A Identify the Requirements Engineering challenges of using agile methods in large-scale systems development.</td>
</tr>
<tr>
<td>RO.1B To explore the state of the art on challenges of developing Safety-Critical Systems in agile environment.</td>
</tr>
<tr>
<td>RO.1C To identify challenges of using agile methods in structured environments.</td>
</tr>
<tr>
<td><strong>G.2: Explore solution space</strong></td>
</tr>
<tr>
<td>RO.2A Examine and critique existing popular frameworks for managing scaled system development.</td>
</tr>
<tr>
<td>RO.2B To propose techniques to solve or overcome some of the identified challenges</td>
</tr>
</tbody>
</table>
1.2.1 Gain insight in current state of RE

The first sub-goal G.1 was set to gain insight into the current RE challenges in the agile industry. The topic for RO.1A explores general challenges of performing RE in large-scale agile systems organisations. This topic was selected with request from participating companies that were in the process of adopting agile methods and thus formed the basis for the subsequent objectives.

Through RO.1A, the role of RE in large-scale agile development was re-emphasised and a range of challenges discovered. Since we aimed for empirical research, it was essential to perform relevant research while also keeping the practitioners interested. Thus, while RO.1A gave a broad scope of challenges, we explored the ones that geared interest for the participating companies and also proved less explored by academia. Without wanting to miss important RE related issues, we explored research on safety in agile development. In so doing, a study on the challenges related to safety-critical and agile development led to the second objective RO.1B. RO.1B aimed to find out which challenges have been identified in research that relate to safety systems in agile environments.

RO.1A also found different scopes of agile adoption in practice giving a hint of coexistence of both agile and plan-driven methods. This finding created a need to investigate the challenge of having both methods in industry. RO.1C based on one case company to explore the coexistence phenomenon. Challenges in this context were sought from the perspectives of the development teams.

1.2.2 Exploring solution space

After gaining insight and understanding our problem space, we aimed to explore the solution space as well. We explored the solution space in three separate ways which aimed to meet two objectives. RO.2A was set to explore the already designed frameworks and find out how, or if at all, they are addressing the identified challenges. Understanding which practices the scaled frameworks recommend for addressing the identified challenges also helped us to understand the challenges a bit more. With views from the practitioners, we also sought to understand whether the frameworks are helping and how it is that the challenges continue to surface.

RO.2B explored available techniques for overcoming (any of) the identified challenges. Through RO.2B, we identified other possible interventions that could be used to address some of the identified challenges. We started with a design and implementation of a tool that showed promising results for addressing some of the challenges. We also started to derive on a taxonomy that could help address coordination concerns in large-scale agile systems’ development.

1.3 Research Methodology

This thesis builds on six studies (Papers A–E), i.e. four empirical studies that adopt a case study approach (Papers A, C, D and E), one secondary study (Paper B), and one tool study (Paper F). The work in this thesis follows an empirical research methodology. It collected, analyzed and evaluated empirical evidence on the phenomena of interest. This section describes the overall
1.3. Research Methodology

The overall research of this thesis was motivated by our industrial partners’ enthusiasm to improve their RE management process. For this reason, the majority of the work was performed as qualitative empirical studies. The collaboration with industry partners allowed us to base our findings on empirical data from practitioners and also validate the relevance of our research questions.

Since software development is carried out by persons or groups in organisations, this makes it a multi-disciplinary area that also includes social boundaries. Thus, for this research, we investigate not only the tools and processes used by the development teams but also their social and cognitive processes. For studying these real-life situations of practitioners in our partner organisations, the use of qualitative approaches became necessary.

The qualitative studies allowed us to get insight into the RE challenges that our industry partners encounter and also derive solution candidates. We also include a literature or secondary study paper that allows us to zoom out of RE to relate to the bigger context of safety, with which many large-scale system development companies are concerned. We elaborate on the qualitative methods used and the secondary study in the next sections.

Figure 1.4: Research process of thesis

Research approach of the thesis, followed by a description of the methods used, namely; a secondary study through mapping study, and case studies.

1.3.1 Research Approach
1.3.2 Case Study Method

This thesis bases on a case study definition given by Runeson et al. 

According to Runeson et al., a case study is an empirical inquiry that draws on multiple sources of evidence to investigate one instance (or a small number of instances) of a contemporary software engineering phenomenon within its real-life context, especially when the boundary between phenomenon and context cannot be clearly specified. Furthermore, case studies provide an in-depth understanding of why and how given phenomenon occur, thus giving opportunities to describe, explore and explain the studied phenomenon. This thesis includes studies that are mainly exploratory (especially Paper A and C), in that they seek new insights and identify useful distinctions that clarify our understanding about agile development and RE processes in industry. Paper C and D endeavour to describe the reasoning behind the identified challenges and identify solution candidates from literature and practitioners. Paper E also uses a case study approach and attempts to derive a taxonomy aimed towards improving the current situation. Paper F introduces and describes a Text-based Requirements system (T-Reqs) that was fronted as a solution to some of the identified challenges. T-Reqs is the open-source version we created based on a working tool at one case company.

Runeson and Host present five steps to performing case study research and these include: (i) designing case study objectives and plan (ii) Defining data collection procedures and protocols (iii) Collecting data on the studied case, (iv) analysing the data and (v) reporting the data. The researcher participated actively in all these steps for the appended studies. For the first step, we had our research questions prepared before starting the study and purposely identified the cases to use in our studies. For all the studies, we chose large-scale systems development companies that have agile development teams. While identifying the companies, we also discussed with the contacts in the different companies on the acceptable data collection procedures.

Case studies can be (i) holistic–with single or multiple case studies is the unit(s) of analysis, (ii) embedded–where one can either have a single case study with many units of analysis or multiple case studies each with multiple units of analysis. This thesis used two of those four described settings; holistic multiple case study for papers A, D and E, and one embedded single case study for paper C. Each setting was used to satisfy the aims we had in each study. For instance, in Paper A the aim was to find out the general challenges of doing RE in agile development companies and thus a multiple case study approach, with four cases of study, was chosen. The cases were all from different domains. This setting allowed us to discover as many challenges as there were, and we could attempt to generalise when we find challenges reoccurring in different domains. The findings of Paper A inspired the study that led to Paper B and C. For paper C, we used a single case study with two units of analysis as we aimed to find the challenges that individual agile teams faced working in a structured environment. The single case was viable since we needed to focus on the same environment setting and the two units of analysis shed more light on the actual situation enabling us to explain the phenomenon. For each of these case studies data was collected through interviews, focus group meetings and workshops where necessary.
1.3. RESEARCH METHODOLOGY

Focus Groups and Workshops  We used workshops as initial data collection instruments for most of our studies (Papers A, B and D) and backed them up with focus group meetings to get more detailed understanding and explanation. We differentiate between workshops and focus groups in that for a workshop, a group of practitioners—generally knowledgeable about agile development and RE—meets to work on creating a defined result jointly while focus groups involved representative stakeholders or experts that were invited to discuss the specific topic under investigation from all relevant perspectives. For instance, in Paper A, the workshops involved company contacts from all participating companies whereas focus groups were conducted at company sites with the representative practitioners of RE and agile development. Focus groups allowed us to dig deeper focusing on one company while workshops allowed us to triangulate our findings. In general, a total of 13 workshops with 79 participants and 7 focus group sessions with 26 participants were conducted. The totals are distributed in the different studies as shown in Table 1.4.

For all the workshops, at least three (3) researchers were always present whereas focus groups sessions were attended by at least two (2) researchers. The researcher was always one of those present and participating in workshops and focus group sessions. Prior to workshops and focus group sessions, a tentative agenda was shared with agreed participants. In many instances, the agenda included a presentation from the practitioners, detailing their experience with RE and agile development (Paper A, C and D), with safety (Paper B) or with islands and boundary-objects (Paper E). We would then present work related to the topic of discussion to ensure a common understanding of the topic. In most cases, this would raise discussion points for which notes would be taken. The researcher would present at some sessions (Paper A, B) and actively take notes while seeking clarifications where necessary at other sessions (e.g. Paper D, E). Follow-up questions and discussions were kept welcome and open. The sessions lasted three (3) hours in most cases (e.g. in Paper A, B, E) and full-day, at Chalmers university premises, in one paper (Paper D). We would summarise the session on one or more slides as a way to ensure we understood the participants’ input correctly.

Interviews  The researcher was actively involved in 18 of the 29 (unique, see Table 1.4) interviews that were conducted in this thesis. Interviews were used in studies leading to papers A, C and D. For Paper C, interviews were the primary source of data since it targeted development teams in particular. Semi-structured interviews, where one or more interviewers interacted with one interviewee based on an interview guide, were also used to collect data. For each interview study, an interview instrument with a structured set of questions was designed prior to the interviews. The interview instrument was created with recommendations from the contact persons in the respective companies. These recommendations allowed the researcher(s) to ask questions that interviewees would understand in their context.

The interviews started with an explanation to the interviewee about the focus of the study to ensure that the same topic was being discussed. During the interviews, the instrument acted as a guide and was kept open in such a way that there was no strict adherence to the structure and flow of the questions. Both the interviewer and the interviewee were free to ask for clarification and
follow-on questions which allowed wide and deep inquiries into the reality of software development for the specific topics of interest. All interviews were audio-recorded with interviewee consent and later transcribed leading to textual analysis.

**Data analysis** For the data analysis, since we were dealing with qualitative data, we relied on a thematic coding approach \[78\]. Since we worked in groups, we had at least two researchers at each study to familiarize themselves with the data collected while highlighting noteworthy statements and assigning codes or labels to each. The researcher was always one of the two researchers at each coding phase. For the interview data, the researcher transcribed the data and performed the initial coding. The codes would then be discussed as a group and iteratively agree on the themes which we would discuss through workshops (Paper A, D, E) or through email and telephone calls (Paper B and C) for validation with the participating cases. Paper F describes T-Reqs which was inspired by a tool already in use in one of the companies as their in-house solution. Together with support from the pioneers of that in-house solution, we created an open-source version of T-Reqs through defining simple templates and scripts. For T-Reqs, we relied on feedback from the company contact to verify and improve its usefulness.

### Table 1.4: Summary of Research Methods

<table>
<thead>
<tr>
<th>Paper</th>
<th>Data Collection</th>
<th>Analysis Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Type</td>
<td>No. of participants</td>
</tr>
<tr>
<td>Paper A</td>
<td>Holistic multi-case</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>5 focus groups</td>
<td>2 workshops</td>
</tr>
<tr>
<td>Paper B</td>
<td>Mapping Study</td>
<td>Mapping study</td>
</tr>
<tr>
<td>Paper C</td>
<td>Embedded single</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>1 focus group</td>
<td>18 interviews</td>
</tr>
<tr>
<td>Paper D(^2)</td>
<td>Holistic multi-case</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>5 focus groups</td>
<td>11 workshops</td>
</tr>
<tr>
<td>Paper E</td>
<td>Holistic multiple</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1 focus group</td>
<td>2 workshops</td>
</tr>
<tr>
<td>Paper F</td>
<td>Holistic single</td>
<td>Tool design</td>
</tr>
</tbody>
</table>

\(^1\) 11 of the 18 interviews were used in Paper A.

\(^2\) Paper D shares focus group, interview and 11 workshop participants with paper A.
1.3.3 Secondary Study

With the growing number of empirical studies in software engineering comes a necessity to construct an objective summary of the available research evidence to aid in decision making and formulation of research questions \[79\]. Using a systematic literature review is one way of obtaining the objective summary. Much as each study included in this thesis has a literature review section, a systematic review provides guidelines to follow while reviewing literature in order to avoid bias and ensure replicability \[80\]. Systematic literature reviews, however, require a considerable amount of effort \[81\].

A systematic mapping study provides a map of the results reported in literature, usually a more coarse overview thus often requires less effort than a systematic literature review \[80\]. The process followed in a mapping study is also systematic but more coarse than a systematic literature review, allowing to process larger numbers of papers.

System development companies are usually subject to standards and regulations, thus they aim for a transparent RE process. Also, with the increased digitalisation and interconnectedness of devices, safety becomes a concern in development. Following the findings from Paper A that identified challenges with safety, a mapping study (Paper B) was conducted to help us draw a map on the current state of affairs in as far as agile development and safety-critical systems development are concerned. The researcher performed the initial document search that gave 1986 documents and was able to reduce them to 69 documents when following the inclusion and exclusion criteria that were defined. Together with two of the other researchers, more studies were excluded giving a final total of 34 documents whose data was analysed and findings presented in Paper D. We aimed to classify and synthesize our findings from industry with those in published empirical studies. With this, we derived a viable research direction for managing requirements in large-scale agile development.

1.3.4 Threats to validity

For this thesis, as with many empirical studies, there are validity threats worth discussing. We consider the four perspectives of validity threats as presented in Runeson and Host \[74\] and in Easterbrook et al. \[75\].

1.3.4.1 Construct validity

Threats to construct validity refer to the relations between the research method and the observations from the study \[74\]. With these threats, the question to answer is: Are the theoretical constructs interpreted and measured correctly? There is a threat that the interpretation of the questions asked at the interviews may be different for the researcher and the interviewees due to the use of different or abstract terms. To minimise this threat, we relied on our company contacts and selected participants who are knowledgeable in the subject of study. On the general scope, participants had to have knowledge on the constructs of agile development and RE. We collected data from multiple sources, including existing literature and different companies in varying domains. This diversity of sources helped us to ensure we got correct results. For the interviews, we shared and discussed the interview guide with the company contact persons in
order to agree on the commonly understood/used terms at the company and also used literature to provide a link between our understanding and that of the interviewees. In cases where the interviewee did not understand the question, we endeavoured to rephrase and give explanations. We also asked interviewees for elaborations in case we got an ambiguous answer. Most of the data collected in interviews was validated in workshops and focus group meetings. These meetings always began with presentations from the company participants and also from one of the researchers. Judging from the presentations that were given by practitioners, we were always confident that the concepts were clearly understood. We also ensured that we had more than one researcher for data collection and analysis in all the studies. To combat the practitioners’ fear to answer asked questions with honesty, we guaranteed anonymity and raw data was only to be used by the researchers. For Paper B in particular, we also calculated inter-rater agreement where there seemed to be some disagreements. This calculation was then followed with discussions, among the researchers, to resolve the disagreements.

1.3.4.2 Internal validity

Internal validity focuses on the research design and whether the results really do follow from the data [75]. For internal validity the question to answer is: Could external factors impact the results of the investigated factors? To minimise this risk, and with permission from the interviewee, we recorded all our interview sessions in order to ensure that each researcher gets the same message at data analysis phase. We also used data triangulation between interviews (Paper A and D), between the units of analysis (Paper C), and between the case companies (Paper A and E). Furthermore, the results of our case studies were discussed in workshops (Paper A, D and E) and at focus group meetings (Paper C). The workshops and focus groups included practitioners from these companies that were already involved in the respective studies. These practitioners were always allowed to discuss their (sometimes different) perspectives on the data we collected, thus increasing our confidence in the data. Through sufficient numbers of interviews and member checking, we made sure that we captured all important concepts in the scope of our inquiries. To avoid a too restricted view on smaller parts of a project or a product, we selected interviewees from different parts of the development. There might, however, still be a selection bias as the interviewees were selected through a convenience sample through our company contacts.

1.3.4.3 External validity

External validity relates to the ability to generalise the results beyond our case studies. By design, the external validity of our studies is low. Hence, generalisation of our findings to different domains or companies might not be possible. Easterbrook [75] notes that qualitative studies aim to understand and explain a given phenomenon rather than generalising. Understanding the investigated phenomenon in one setting may help to understand other situations. For instance, in Paper A, we designed our study to identify common challenges across participating companies. Thus, our research method does not support any deep reasoning about differences between companies, domains, and
market positions. However, given that we found similar themes in all cases, we expect that these apply similarly to other companies or projects in large-scale systems engineering. By analysing data gathered from different companies with different characteristics, we believe that we have got results that can be extended in the future and were able to identify relevant categories that are applicable in other contexts.

1.4.4 Reliability

Threats to reliability refer to level to which the study results are dependent on specific researchers. We limit reliability threats by improving the interview instruments in multiple iterations and by conducting interviews in pairs of two researchers. Also, at least two researchers were involved in the focus groups and workshops in order to reduce the impact of subjectivity. We have continued involvement with our case companies and therefore a mutual trust among the parties exists. The data analysis was discussed and refined among the authors in several iterations. The results were discussed with the participating companies and also compared results obtained to available literature. We also tried to describe our data collection and analysis procedures and shared the instruments used for data collection in the studies conducted. The potential solutions proposed are based on our reasoning, claims in related work (that these solutions help with a specific challenge), and on discussions with the case companies. We have not applied the solutions from the literature in the case companies, or solutions suggested by one company in further companies. Further validation of the collected solutions is needed.

1.4 Research Synthesis

We summarise the main results and contributions of this thesis per research objective and reported study. The detailed descriptions of the results for each study can be found in the respective paper (Chapter 2–7).

1.4.1 Paper A: RE Challenges in Large-Scale Agile

RO.1A: To identify the RE challenges of using agile methods in large-scale systems development.

There is a substantial amount of research on agile methods from their definition to their adoption, even at scale. Research on RE in agile environments is, however, limited. Although there is available literature exploring challenges of agile RE, none has the context of scaled system development companies. Motivated by the limited empirical studies on RE challenges particular to large-scale agile systems development, we conducted the study leading to Paper A which also formed the basis for the subsequent studies. Paper A contributes to our first goal (G.1 – establishing the current use of RE) while meeting our first objective RO.1A, based on the following research questions:

RQ1A.a.: What are possible scopes of applying agile methods in large-scale system development?
RQ1A.b.: How is the role of requirements characterized in large-scale agile system development?

RQ1A.c.: Which requirements related challenges exist in large-scale agile system development?

Main findings: With the first question, RQ.1A.a, we aimed to establish the level to which our four case companies were agile. We found that the participating companies were in the process of adopting agility and thus had adopted agility in differing levels. Whereas one had adopted agile methods for the whole product development process, several of the companies had agile adoption only in the agile teams. This finding meant that the system-level was still following a plan-driven approach, which also gave slight differences in the challenges faced.

RQ.1A.b helped us confirm the importance of requirements, even in the face of agile methods development. We find that requirements are seen as an order while the teams prefer to work with user stories that are portrayed as goals and give them more autonomy. RE is still crucial for system development, and the study revealed that large-scale systems companies are struggling to perform RE in agile development to the level they are used to while they were using waterfall/traditional methods.

However, irrespective of the level of adoption, all companies exhibited challenges (RQ.1A.c) that were put under two particular groups: Shared understanding of User Value and Building and Maintaining System understanding. In regard to user value, most challenges were coming from teams struggling to understand customer value, writing meaningful user stories and feedback and requirements clarification from the user stories or features they develop. For system understanding, the challenges faced relate to informing and synchronising between teams, creating and maintaining traces, insufficient tests and user stories, agile tool chain establishment and, coming more from the traditional foundation, there was a gap between plan-driven development and agile development. Generally, findings demonstrate how system development is struggling with RE, and this has now become an essential topic for practitioners and researchers alike. Existing literature concerning RE in agile development does not provide approaches for both user and system requirements specification in an agile environment.

Potential application of the results: Paper A contributes a map of RE related challenges in scaled agile system development. Practitioners find this map useful to plan their adoption of agile methods as well as check their process improvement since it allows to avoid over-optimizing one aspect while negatively affecting another aspect. We hope that researchers benefit from our overview of challenges when conducting related studies. To follow up on this study, we conducted an independent study on challenges specific to safety (Paper B), since the companies having to deal with safety requirements seemed to show some differences and one following up on the teams being agile while system-level is not (Paper C).
1.4.2 Paper B: Safety-Critical Systems in Agile

RO.1B: To explore the state of art on the challenges of developing SCS in an agile environment.

Agile methods have been criticised for neglecting upfront requirements and extensive documentation [30] which are some of the defining features of safety systems’ development. This criticism has thus created thinking that agile methods are not suitable for Safety-Critical Systems (SCS) development. Upon finding a challenge of developing SCS in agile development among companies dealing with safety systems in Paper A, we sought to understand what challenges have been identified from published empirical studies. Since we did not find a study with a comprehensive and recent overview to help us understand the context, we ventured to explore the domain through a mapping study. We included only published empirical studies from 2001 to 2017. Paper B meets this objective by answering the following research questions:

RQ1B.a.: What research exists about agile development of Safety-Critical Systems?

RQ1B.b.: What are the key benefits of applying agile methods and practices in SCS development?

RQ1B.c.: What challenges exist with agile development of SCS?

RQ1B.d.: What solution candidates (e.g. principles and practices) promise to address challenges with respect to agile development of SCS?

Main findings: The results obtained showed that the focus on SCS has changed considerably over the years from being positive in the beginning, i.e. identifying benefits, to the recent papers discussing more of solutions to overcome identified challenges. The reported benefits were in agreement with the ones reported in the non-safety development domains, for instance better test cases, improved quality and improved safety culture.

Apart from not trusting agile methods for safety development, the challenges faced in development of SCS using agile methods are more geared towards the certification and assurance requirements of safety systems. Assurance requirements, for instance, involve many stakeholders who come from different domains. Managing communication between such numbers proved challenging in the agile context where documentation is not used for communication or even given that much attention. Also, standards were written with a waterfall mindset which favour effectiveness of waterfall methods over the flexibility of agile methods. Obviously there were challenges with upfront planning, lack of documentation focus and the trade-off of flexibility vs safety which have always been a cause for the skepticism. Generally, safety requirements call for well-structured processes, and heavy documentation that is not clearly supported by agile development and agile teams find it hard and cumbersome to balance speed and flexibility with the need for documentation.

The solutions proposed relate to using much the same practices as in the non-safety development case but also include discussions relating to safety for instance in the daily meetings and sprint reviews. It was also encouraged to
have safety experts as part of the team so that teams always have safety in their mindset(s). We also notice that some practices that have existed even in the traditional methods are highly recommended, e.g. setting high coding standards and relying on standard operating procedures to improve quality management.

**Potential application of results:** The results provide for potential generalisation to other areas. We know the research that exists and that allows us and the companies to systematically search for a setup on being large-scale agile using a map of these challenges. The results also indicate that the challenges faced in SCS development come from the need for structured processes that the standards impose. With the standards in mind, it becomes necessary for companies adopting agile methods to work on a coexistence plan that should have both practices in proper use. Results also provide a starting point for enabling us to accumulate more knowledge on which solutions could help. So future research can build on it rather than replicate it.

### 1.4.3 Paper C: Agile Islands in a Waterfall

**RO.1C:** To identify the challenges of using agile methods in structured environments.

Findings from paper A (and partly from paper B) show that there is a challenge of dealing with the gap between plan-driven and agile development and having ‘agile islands in a waterfall’ organisation. Paper A findings also indicate that some companies have agile adoption only in the development teams an occurring commonly noted in some studies but not yet explored further. Paper C sought to follow-up on that finding to reveal the challenges that agile teams in such traditional structures were facing. We explore the challenges from perspectives of two of the agile teams in one large-scale company. The research objective *RO.1C* was met by answering the following research questions:

**RQ1C.a.:** What are the perceived challenges when combining plan-driven and agile paradigms in large-scale systems engineering?

**RQ1C.b.:** What mitigation strategies exist for the challenges identified?

**Main findings:** The findings of this paper indicate a variation in challenges (*RQ1C.a*) faced for departments of the same company. The difference in challenges stems from the departments’ different ways of working with agile development methods. We find that whereas both departments have adopted agile methods relating to Scrum, they have customised them to their specific needs and thus modified a few roles and practices that bring in the difference. In one department for instance, the role of product owner was not defined although requirement prioritisation and coordination seemed to be going on well. Also, the requirements management tools used in both departments differed considerably although for both departments developers had no access to the high-level or system requirements. In that context, we found challenges common to both departments, e.g. development teams not being aware of the high-level requirements, function owners over exposed to change requests
and traceability issues mostly stemming from difference in tools and unclear responsibilities in such environments.

We also found challenges unique to each department. At one department, there was less focus on control and more ambition to facilitate autonomy of agile teams. This resulted in practices where, for example, anyone on the team could write a user story to the backlog. It was found that this could lead to temporal inconsistencies, between the user stories and implementation, that surfaced and had to be fixed during testing. At the other department, such challenges were not as evident as the challenge of not knowing what effects a change in one requirement could have on dependent units. Since developers in this department were working with a central platform that is the foundation for the work of several other departments, they felt the lack the complete picture of the function was a more pressing challenge.

Strategies to mitigate the identified challenges were obtained both from the participants and also from literature. On the one hand, participants felt it necessary to have (system) testers as part of the team, update requirements based on learning from sprint, create proper channels for writing user stories while also explicitly stating which user story must be traced. Literature, on the other hand, recommends cross-functional teams to manage requirements updates, increasing understanding of processes and roles in development and bringing testers closer to the requirement owner. However, these strategies remain abstract and empirical research on their effectiveness is currently lacking.

Potential application of results: The results suggest a need for a holistic company-wide approach to agile adoption and development to overcome some of the challenges. Although there exist studies that mention the possibility of different ways of working for sections of the same company, this is the only study, that we are aware of, that clearly documents it while providing the proof of what challenges could ensue. From this study, we also observe that while it might not be possible to have all parts of the company agile nor be desirable to have all parts of the company plan-driven, if different approaches must co-exist, one should find a way to integrate them. Future research may need to show how this integration can be achieved.

1.4.4 Paper D: RE practices in large-scale Agile

RO.1A: To identify the RE challenges of using agile methods in large-scale systems development and

RO.2A: Examine existing scaled frameworks

Many frameworks have been proposed to try to overcome the challenge of agile development at scale. However, their usefulness or ability to solve the challenges has not been explored as elaborated in Section 1.1.3 With Paper D, we explore what solutions two of the popular frameworks, SAFe and LeSS, offer in relation to our identified challenges. Paper D contributes to research objective RO.1A and satisfies RO.2A by answering the following questions:

RQ.2A.a: How pervasive are agile methods in large-scale system development?
RQ.2A.b: Which requirements-related challenges exist in large-scale agile system development?

RQ.2A.c: Which approaches have been proposed in popular literature and which approaches are used by practitioners to address the challenges identified in RQ.2A.b?

Main findings: Paper D reports on a three-year exploration that extends Paper A with an almost exhaustive catalogue of RE challenges from the participation of seven companies. Thus RQ.2A.a re-affirms that agile adoption still exists in different phases in large-scaled systems development companies, with the three more companies in perspective. We identify new challenges (RQ.2A.b) to those identified in Paper A and rearrange them to six categories which we briefly elaborate in the following:

1) The first category, *Build and maintain shared understanding of customer value*, comes from the core strength of agile methods – managing customer value. Challenges here include bridging gap to customer and building long-lasting customer knowledge. For these large-scale companies, the distance between customers and development is large, and it is difficult to break the features into meaningful packages that have customer value and can be delivered incrementally. The distance also makes the customer role unclear at scale.

2) For the second category, *support change and evolution*, we find challenges relating to managing requirements updates, management of experimental requirements, synchronisation of development and requirements re-use. We find that updates are manually done, leading to inconsistencies which are expensive to remove. This indicates that facilities for updating system requirements based on agile learning are currently missing.

3) *Build and maintain shared understanding about system* is our third category. With agile methods’ focus on value, the system requirements knowledge is underrepresented. We find the need for documentation to complement test cases and user stories, which are common practices in agile development, as they are not sufficient for system understanding. Furthermore, the big picture of the system is not captured since teams focus more on features or components and thus, the creation and maintenance of traces becomes complex as well.

4) The fourth category *Representation of Requirements Knowledge*, denotes the shared responsibility of requirements knowledge. In particular, it is hard to manage the various levels of development in large-scale while giving meaningful decomposition of requirements in agile development. We also find that the current tools being used are not fit for agile development as they do not provide the flexibility needed to accommodate different representations of requirements that teams or individuals could have. Additionally, it is difficult to establish consistent requirements quality and also align or establish thresholds for quality requirements.

5) Our fifth challenge category, *process aspects*, relates to the process of working with requirements. We find challenges relating to prioritisation of distributed functionality, managing requirements completeness and consistency. We also find the common challenge of balancing between time-to-market and quality of the product.

6) *Organizational Aspects* formed our sixth and final category. For these
large-scale systems engineering companies, the overall organisation in which RE is practiced plays a vital role as well. Inherently, these companies perform (some) long-term planning, especially for facilities. Our challenges in this category relate to bridging between such system-level planning and agile work in software teams, planning of integrated system testing, managing research and pre-development, and identifying impacts on critical infrastructure in good time.

Practitioners made suggestions towards solutions for some of the identified challenges (RQ.2A.c). For instance, many of the practitioners agreed that it is a good practice to have the teams and PO update the requirements. In this way, the gap between plan-driven and agile development would be bridged, and the tooling not fit for purpose challenge addressed. They also shared a recommendation to move from project-focused development to product-focused development in order to create and maintain traces as well as encourage re-use of requirements. Here, practitioners also mentioned the Text-based Requirements system (T-Reqs) as a useful tool for updating and managing experimental requirements. Paper D gives more details on these challenges and solutions, while we expend on TReqs in Paper F.

A mapping of the challenges to the solutions provided by SAFe and LeSS frameworks (RQ.2A.c) revealed that these frameworks have in some cases concrete practices while in other cases, there is none. In relation to Build and maintain shared understanding of customer value category, we find that several practices have been proposed. We find solutions relating to concrete practices, (e.g. frequent demos, sprint review bazars, use models, continuous improvement, retrospectives). We also find more abstract guidelines. For instance SAFe describes techniques such as combining “weighted shortest job first”, “portfolio backlog”, and “program kan-ban” to support cross-cutting initiatives towards prioritization. It also advocates for a combination of other practices as elaborated in Paper D. However, there are no clear guidelines on how to combine all these suggestions into one coordinated process. The same can be said about many of the recommendations from LeSS framework. It is our understanding that such combinations would require specialised tools, or even just customised tools which do not yet exist.

Results from analysis of popular frameworks show a lack of concrete advice to manage some of the challenges (coming from the different categories) while for many others, the solutions from these frameworks are relatively underrepresented.

**Potential application of results:** The Paper presents a catalogue of RE challenges and their potential solutions both from practitioners and also recommendations from popular frameworks. This catalogue would help inform practitioners on which solutions they already have available through the scaled frameworks and what practitioners in similar settings have used to manage their situations. Additionally, in order to mitigate the identified challenges, we encourage future research to not only focus on producing further practices but also evaluate the existing and proposed solutions in large-scale agile settings.
1.4.5 Paper E: Charting coordination needs

**RO.2B:** To propose techniques to solve or overcome some of the identified challenges

Current research explored hybrid development in terms of how development methods are combined by teams during development, for instance, using a mixture of traditional methods with agile methods [38]. What does that mean for large-scale companies working on different projects with dependencies? Our previous studies showed issues with communication and documentation, which were affecting coordination between teams and high-level development. While motivated by the results of paper C and those leading to Paper D, in paper E we set out to search for solutions for identified challenges. We started with generalising the problem of agile islands to the whole system development and thus the whole organisation. We contribute to research objective *RO.2B* by answering the following research questions in paper E:

**RQ2B.a:** Which agile islands are repeatedly encountered in large-scale agile contexts?

**RQ2B.b:** Which boundary objects are repeatedly encountered in large-scale agile contexts?

**Main findings:** Our findings to *RQ2B.a* show that there are occurrences of varying methods being used in the companies but not only among those using agile methods but also non-agile teams. We thus started to think in terms of methodological islands. While evaluating which islands do exist, we realised that the level of abstraction for the island differed and these abstractions were extracted and grouped to give the drivers for methodological islands.

We identified three groups of methodological islands which occur on different levels in the company. At the very high level, we identified external *organisations* that companies have to work with. These include, for instance, suppliers, regulators and customers, all of which come with different ways of working that have to be accommodated. The next two groups come from within the companies. For companies using SAFe framework, agile release trains (teams of agile teams) exist, and companies have to coordinate tasks between the different release trains. Similarly, for companies without SAFe, departments still exist each with its teams. As has always been, product development can span several departments (e.g. marketing, hardware, software departments) and thus several disciplines in the company. Since departments and release trains, for instance, are a group of different teams, at the intermediate level we identified *groups of teams* as another category of methodological islands. At the lowest level in the company, we have the *individual teams*, for instance, component teams or integration teams, which also display different methodologies.

We identified drivers for such methodological islands as process-, business- and technology-related factors. Companies are business-oriented and thus have to watch different factors which are best addressed through having departments. For the large scale, some companies or departments are also distributed to different geographical zones which brings in time and cultural differences. This distribution creates different methods for the affected departments, thus the *business-related* drivers. *Process-related* drivers describe a mixture of
development methods (SAFe \[^{82}\], V-Model \[^{83}\], Scrum \[^{28}\]) and whether a company mainly works based on projects, or whether significant workflows in the continuous development of a platform. The technology-related drivers come from the systems being developed. Here, methodological islands will emerge depending on whether they are dealing with the same architectural decomposition, systems disciplines, platform and product-line, and the required time-scale of commitment.

The boundary objects related to the methodological islands were also identified and categorised depending on different viewpoints of the islands. Typically, for large-scale, before development commences, contracts, roadmaps and plans are created for proper project or product management. We categorised these as planning boundary objects. We further identified boundary objects that relate to tasks in the development effort, e.g. user stories and other backlog items as task boundary objects. Those that are concerned with technological aspects (e.g. tests or architectural objects) of the system being developed were categorised as technology boundary objects. Regulation and standards boundary objects are used to ensure that the company complies with regulations and standards through safety cases for instance. The regulations come with a demand for process definition and thus documentation which is achieved through process boundary objects like SAFe documentation. Boundary objects relating to the final customers’ product were categorised as product description boundary objects and include, e.g., the technical documentation for the customer and the variability model. Trace links are a special category, as they represent the relationships between artefacts.

**Potential application of results:** Paper E presents a catalogue of methodological islands and the boundary objects between them. Our catalogue proved useful when discussing potential process improvements with companies as it gives them a mindset of planning their coordination efforts in relation to their distances and particular needs. So, this catalogue comes in as a starting point for a technique that can be used to identify gaps or distance in islands and also plan for possible intervention before it becomes costly. This catalogue for boundary objects is not exhaustive and future research could pursue creating a model or taxonomy that defines this technique fully. Also, thinking about tools as boundary objects is a viable next step.

### 1.4.6 Paper F: Tool support for managing requirements

**RO.2B:** To propose techniques to solve or overcome some of the identified challenges

We approached the other part of research objective *RO.2B* through developing the T-Reqs tool. The tool design was motivated by Paper A, C, D that reflect findings on the challenge of agile teams not being aware of or able to update the system requirements. Agile methods provide recommended practices which also need tools support to make them effective. To this end, requirements tools are abundant. However, most existing tools for requirements management do not support the autonomy of development teams, e.g., giving developers ability to view, and possibly update the requirements at system-level. We recommend


Table 1.5: Summary of challenges relating to the ‘gap’

<table>
<thead>
<tr>
<th>Category</th>
<th>Related Challenge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organisational</td>
<td>Lack of trust in agile methods</td>
</tr>
<tr>
<td>Organisational</td>
<td>No time for invention and planning</td>
</tr>
<tr>
<td>Organisational</td>
<td>Hard to plan tests based on requirements</td>
</tr>
<tr>
<td>Process issues</td>
<td>Prioritisation of distributed functionality</td>
</tr>
<tr>
<td>Process issues</td>
<td>Managing different representations</td>
</tr>
<tr>
<td>Technology issues</td>
<td>Tooling not fit for purpose</td>
</tr>
<tr>
<td>Technology issues</td>
<td>Lack of overall system knowledge</td>
</tr>
<tr>
<td>Technology issues</td>
<td>Managing traceability</td>
</tr>
<tr>
<td>Technology issues</td>
<td>Updating requirements</td>
</tr>
<tr>
<td>Technology issues</td>
<td>Synchronisation of development</td>
</tr>
</tbody>
</table>

customisable tooling to address the gap between system-level and agile teams. With paper F, a new tooling concept is illustrated.

T-Reqs is a tool solution that gives the developers autonomy and ability to make changes to the system requirements when needed. It is text-based and works with Git version control system to enable agile teams to propose changes to the high-level requirements at system level. This tool is presented in Chapter 7 where it is further elaborated in a technical report.

**Potential application** The tool is applied in the development in one company with a few modifications. The tool works well with reviews and allows the developers to use it within their usual development environments and yet still maintain traceability. It, however, still has artefact (requirement, test case or user story) Identity (ID) generation and more ideas to enhance it with sophisticated capabilities such as models that will not make it more complicated are welcome. The tool highlights the text-based view on requirements and future research could explore which domains such specification works best and if it can be generalised.

### 1.5 Summary of Results

The appended papers each contribute incrementally to the thesis goal of providing an empirical investigation into the challenges and solution candidates of performing effective RE at scale and in systems development. In this section, we summarise the contributions of this thesis as per the sub-goals of the thesis.

#### 1.5.1 Current status of RE (G1)

Findings re-affirm that RE in system development is still as important as in agile development as it is or has been in traditional development. However, the team focus on user stories that come with agile development has added on to the challenges of RE in systems development despite solving, e.g. a few previously known challenges. We explored the challenges of developing in a large-scale agile environment from three ‘angles’ in order to meet this goal. We began with the general large-scale organisation (RO.1A) that reveals challenges
relating to user value and system understanding (Paper A). These findings in paper A are later expanded into six categories in Paper D. This thesis focuses on challenges that relate to system understanding. The studies addressing RO.1B and RO.1C explore system understanding yet further. The second ‘angle’ is the safety-critical domain, RO.1B. This research (Paper B) reveals that challenges in the safety domain stem from the recommendations provided by the standards and regulations, that companies have to fulfil. We find challenges relating to upfront, just-in-time, and long term or infrastructural aspects. However, the trade-off between upfront analysis and just-in-time development is still much of a pain point that is under researched. The third and final setting is the agile teams in a waterfall environment, RO.1C. This study (Paper C) highlights the challenge of the gap between system-level requirements and agile teams development. At the system-level, requirements are more plan-driven while in the development teams, the pace of change of requirements is easily accommodated. It is a challenge to balance the agility of teams and system-level information needs in large-scale agile system development. This challenge is a reoccurring challenge in all the studies and appears to be affected by the entire process of the organisation including business, process and technological aspects. Here we denote the challenges in those categories that also relate to or sprout from that major challenge and are also reported in our studies. Table 1.5 presents the visualisation.

**Organisational issues** In this category, challenges relating to organisational discipline are discussed. Agile at scale was received with skepticism from many project and/or product managers as they lacked the trust in agile methods since agile methods do not give clear guidelines for project monitoring and control. Also, large-scale organisations commonly invest much time for inventions and planning. However, with the introduction of sprints, it is not clear at what level to do it as it would slow down development if given to developers yet at system-level it would mean extensive documentation and handover not recommended in agile development. Test planning (plan V & V based on requirements) is another organisational issue that is affected by agility at scale. The agile teams with their product owners do not perform validation plans which are done by the system managers. This introduces a gap between the agile and system testing team. Also the rate of change of requirements could warrant an update to the testing infrastructure. However, the current setting does not give testers the ability to monitor changes and thus have a head-start in improving the test environment.

**Process issues** Here, we group challenges from the development process that relate to the gap. Prioritisation of distributed functionality is suffering at the gap since teams choose to work with simple tasks leaving out the high priority tasks with the excuse of not enough time to implement. This act frustrates system development as it becomes hard to follow development. Additionally, teams want to tailor requirements to their contexts but there is no support for managing different representations of requirements. At the same time, system-level development aims to keep artefacts consistent and manageable. This also brings in the tooling not fit for purpose challenge. Current tools limit developers’ access to system-level requirements making updating of requirements slow and
cumbersome. Also, since requirements are usually defined at the beginning of development, there is no obvious way to update them.

**Technology issues** In this category, both process and organisational aspects are at play, thus issues relating to technologies used. At the higher end, the organisations choose the technologies to use and at the lower end, the processes depend on (are driven by) the technologies used. Teams develop components that must fit into the full system but lack the overall system knowledge. They thus might sub-optimise with regard to the bigger system as they are thinking of component quality. This challenge can be motivated by the organisation improving its structures which should help the developers become more proactive than before. It is hard to trace user stories on development level to requirements on system-level since developers are also not motivated to create such traces. This challenge is also deepened by the usage of different tools at the development and system level. In the end, the update to requirements also suffers developers work with tools different from those at the system level. With large-scale development, there are usually many levels of requirements as they are decomposed making synchronisation of development hard as channelling the right information from system level down to the developers becomes time consuming and difficult.

### 1.5.2 Exploring solution space (G2)

The solution space is explored in three ways as well. First, we analyse two popular frameworks (SAFe and LeSS) to identify the solutions they offer for the identified challenges. In relation to balancing team agility and system-level information, scaled frameworks have recommended practices. However, there is no empirical proof of which we are aware. We find a lack of guidance on the tools to use, and the proposed practices are not concrete per se. With this finding, we explore alternatives for solutions. Second we start to recommend with a textual tool (T-Reqs) based on Git that allows the developers to propose changes to the high-level requirements. Lastly, we come up with a catalogue of methodological islands and boundary object types that practitioners should recognise in their organisations to help them try to address the coordination challenges between teams and system-level works. The proposed solution relies on artefacts, specifically boundary objects. In view of this aspect, T-Reqs is a boundary object that can also address the coordination needs of teams in large-systems’ development. We recognise process, organisational and technical issues which relate to RE at scale both in the problem space and in the solution space. Our findings in the solution space imply that large-scale companies’ RE needs for agile system development are driven by choices made at all those (process, organisation, and technical) levels. Thus, understanding the factors at play in each of the levels could help address the ‘gap’ challenge.

### 1.6 Discussion

Just as the technology-, process- and business-related (or organisational) aspects drive the occurrence of methodological islands, the current state of RE
challenges has also been grouped in these categories allowing us to understand the challenges better in relation to the solution space. For organisations that have a set organisation structure and defined standards and processes to follow within the organisation, this grouping starts to appear at different levels. At the highest level is the business-related aspects, with processes- and technology-related aspects as subsets. The processes and technologies used are determined by the organisation and thus differences of methods used in teams are easily identified. In this way, organisations only take up processes if they have explored the different ways such processes can be applied to their contexts and thus expect less surprising challenges. In line with recommendations to tailor agile methods to their contexts \cite{84}, we further emphasis a company driven adoption.

This thesis expounds on the challenge of a gap between development and system-level requirements. The gap between agile teams and system-level development has been addressed through hybrid development in recent studies. Hybrid development in the most straightforward form consists of practices from different agile methods and also from traditional development methods. We have seen that this form of development has become the current norm for many companies, big and small alike \cite{36}. For large-scale system development, however, hybrid development goes beyond the combination of different methods for development to having two separate sections in the organisation, one working in a plan-driven (traditional) and another working in an agile manner.

Results on the challenges arising from coexistence of traditional and agile methods reveal, e.g., inconsistencies between the implemented requirements and the requirements being tested (test planning), which comes from the development teams not being aware of the high-level requirements. These results are in agreement with, e.g., Kusters et al. \cite{39} who identify a ‘lack of linkage of the iterative development process to the test process’. This thesis identifies many challenges in this gap (see Section 1.5 In light of such challenges, the success of agile development at scale becomes questionable. Rolland et al. \cite{6} while challenging assumptions of agile development at scale, posed the same question when they examined the concept of self-organising teams at scale. Although not focused on RE in system development, they pose this question “Are there models of organizing the development process that can grant team-level autonomy and still ensure efficient inter-team coordination?” Dealing with safety-critical components of the software and a solid foundation on waterfall methods limits the autonomy that teams can have at scale. We question how far team autonomy can be stretched in that context.

Additionally, our findings show that requirements change so fast during development that it is hard to track the change in dependencies between requirements among teams. At scale these challenges change form and become essential to address \cite{6}. In the systems engineering context with parallel development of hardware and software, these challenges become critical, especially when we add the RE context as then the documentation and entire validation process could be terribly slowed. Struggles to find solutions for such challenges are still on-going.

Kuusinen et al. \cite{40} recommend strategies that include convincing management to change their mindset about agile methods. Relating to our discussions in interviews and workshops, managers seem to be convinced but still challenges
exist since, given the scale and related constraints, it is hard to be fully agile. We believe that buy-in to agile development at scale has to span to the known ‘non-agile’ departments like hardware departments as well. This thesis presents two fundamental attempts to solving the identified challenges. Coming from an RE perspective, development is turned into a communication problem and eventually into a coordination problem when we talk of software development among 6 or more teams. At that level, good strategies for coordination of development need to be devised. T-Reqs and the concept of Boundary Objects and Methodological Islands (BOMI) introduced in this thesis present ideas towards potentially working solutions.

With T-Reqs, the organisations do not change their infrastructure as such. The known rules and standards of development still apply with the flexibility of allowing the developers propose changes and work directly with the requirements. In this way, requirements updating challenges are reduced. Some studies have proposed having cross-functional teams \cite{54,85} to help in updating requirements across the ‘gap’. These solutions are applicable for intra-team coordination and communication, and become problematic across teams. T-Reqs ensures coordination across teams, and requirements are always to the most recent version across teams. In addition, the BOMI concept allows for an agile or autonomous way of identifying communication need and thus creating it. This suggestion relates to the practice of ad-hoc meetings \cite{60} which was found effective. Boundary objects as artefacts can come up whenever they are needed and depending on the development aims, can be maintained or discarded afterwards. We, however, aim for reusable boundary objects to reduce workload and also carry over knowledge from previous development.

In short, T-Reqs and BOMI potentially address some of the identified challenges. T-Reqs potentially addresses the challenge of lack of agile tooling (tooling not fit for purpose), managing requirements updates during development and traceability management. BOMI concept is good for advising processes to synchronise development, managing different representations and traceability management. It is almost impossible to define a generic solution and thus we propose these two concepts as they can be customised to the particular company needs. Although they may appear constraining, these solution candidates offer teams an opportunity to be autonomous and efficient while working with system-level information that is process-driven. These two solution proposals call for a full organisational buy-in to agile development.

In the context of RE in large-scale agile systems development, whereas agile RE aims at using agile practices to do RE, we argue for RE for agile development (RE4Agile) where we perform RE to support agile development. This means that the known methods of doing RE would still hold. However, they would not be done one off as before and would also be done incrementally and concurrent to development. This thesis argues that, for large-scale systems development organisations, an approach that takes advantage of the speed and change handling of agile development while also building on the coordination provided by traditional methods would be more adequate. We thus recommend that such large-systems companies to aim for an informed hybrid approach that takes the best of both worlds.
1.7 Conclusions and Future Work

This research aimed to uncover RE challenges and solution candidates for the use of agile development in systems development companies. Through use of a series of qualitative empirical studies, we contribute a catalogue of RE challenges in large-scale agile systems development. We found challenges relating to maintenance and long-term product support which agile methods tend to ignore. Future research could explore that direction a bit more. This thesis dwells on the challenge of system understanding to highlight the challenges of interaction between agile teams and system-level requirements. While companies continue to struggle with the gap, we see a need for future research specifically aiming at investigation of trade-offs between effort done upfront and just-in-time. We also advocate for guidelines on how to shift more effort into just-in-time analysis as we aim towards RE for Agile development.

Through exploration of solution candidates, this thesis also provides a first attempt to skip the divide between upfront and just-in-time analysis. While uncovering the solution candidates, we noticed practices recommended in literature but did not yet find empirical evidence confirming their usefulness for large-scale systems development. We thus encourage future research to not only produce more practices to solve open challenges, but also focus on evaluation of existing large-scale agile system proposals from a requirements perspective. We also explored solutions offered by two of the popular large-scale frameworks, SAFe and LeSS. We found that whereas they have recommended practices, these are still abstract and not concrete enough for the practitioners to implement with certainty. Perhaps studies detailing the successful implementation of these frameworks while elaborating how the proposed practices were met could help in that aspect.

We introduce, T-Reqs, a custom made solution in one of the companies which we, together with its pioneers, created an open-source version of. Although the tool has not yet been tried in other environments or contexts, it is a first step towards addressing many of the identified challenges for large-scale systems development. We aim to continue adding functionality and exploring its usage and thus welcome contributions towards making the open source tool more usable.

We started to explore practical ways to address knowledge management in large-scale agile systems development. For that, we charted a landscape of methodological islands and boundary objects (BOMI) which practitioners found useful for discussing potential process and tool improvements. We are currently working on translating the landscape into a model that can benefit practitioners. Future research could use our landscape to prioritise and scope knowledge management needs. Additionally, a quantitative survey could provide information on which boundary objects and methodological islands are most frequent. With these contributions in the solution space, we provide a sketch of promising approaches, e.g. through agile tools and BOMIs, on how these challenges could be approached.
Chapter 2

Paper A

Requirements Engineering Challenges in Large-Scale Agile System Development

R. Kasauli, G. Liebel, E. Knauss, S. Gopakumar, and B. Kanagwa

Abstract

Motivated by their success in software development, companies implement agile methods and their practices increasingly for software-intense, large products, such as cars, telecommunication infrastructure, and embedded systems. Such systems are usually subject to safety and regulative concerns as well as different development cycles of hardware and software. Consequently, requirements engineering involves upfront and detailed analysis, which can be at odds with agile (software) development. In this paper, we present results from a multiple case study with two car manufacturers, a telecommunications company, and a technology company that are on the journey to introduce organization wide continuous integration and continuous delivery to customers. Based on 20 qualitative interviews, 5 focus groups, and 2 cross-company workshops, we discuss possible scopes of agile methods within system development, the consequences this has on the role of requirements, and the challenges that arise from the interplay of requirements engineering and agile methods in large-scale system development. These relate in particular to communicating and managing knowledge about a) customer value and b) the system under development. We conclude that better alignment of a holistic requirements model with agile development practices promises rich gains in development speed, flexibility, and overall quality of software and systems.
2.1 Introduction

Despite wide critic, agile approaches have significantly contributed to the way software is developed [30]. While initially focused on small teams [7,30,86,87], success stories have led to their application at large scale [8,9,42] and in system development [9,10,43], an environment that is characterized by long lead times [10] and stable, sequential engineering practices [44]. In these environments, new challenges arise, especially with respect to managing requirements [88] and companies struggle to implement efficient requirements engineering (RE) [14,89,90]. Existing works on agile RE (e.g., [19,23,54]), mostly focus on proposing new approaches, practices, and artifacts [16]. There is however a lack of empirical studies that investigate the phenomenon of RE in relation to agile methods in the domain of large-scale system development [15,16,23].

In this paper we report **RE related challenges of large-scale agile system development** i.e. the development of a product consisting of software, hardware and potentially mechatronic components that includes more than 6 development teams [8] and is aligned with agile principles [30]. Through a multiple case study of four large-scale system development cases, based on 5 focus groups, 2 cross-company workshops and 20 semi-structured interviews, we answer the following three research questions from an RE perspective:

**RQ1:** What are possible scopes of applying agile methods in large-scale system development? We aim to better understand the general context of potential challenges in industry.

**RQ2:** How is the role of requirements characterized in large-scale agile system development? With companies turning towards agile software development methods or even scaling up agile practices such as continuous integration to system level, the role of requirements is far from clear. Thus, we examine different situations in which requirements are used in agile software development.

**RQ3:** Which requirements related challenges exist in large-scale agile system development? Building on RQ1 and RQ2 results, we then investigate challenges and implications to RE.

The contribution of this paper is a report of real-world RE challenges related to applying agile development in large-scale systems. These challenges are effectively hindering a faster and more sustainable development of software. We find challenges with respect to the setup of agility in large-scale system development and with communicating knowledge between different parts of organizations. In particular, there is a challenge of distributing and breaking down knowledge about customer value and about building and maintaining system understanding. We are also highlighting the need for systematic approaches to engineering requirements. Thus, we hope that our work helps to establish RE practices that better support agility within large-scale system development.

2.2 Background and Related Work

Agile methods like Scrum and XP are being adopted in large-scale system development companies [42], even though they were originally intended for use on a small scale [7,86,87]. Existing work on this topic shows that companies successfully adopt agile methods, but that several challenges remain. In
2.2. BACKGROUND AND RELATED WORK

A survey with 13 organizations in 8 European countries and 35 individual projects on the adoption of XP and Scrum, Salo and Abrahamsson \[42\] report successful adoption of these methods and appreciation among practitioners. Lindvall et al. \[13\] study the potential of adopting agile methods with ABB, DaimlerChrysler, Motorola, and Nokia. The authors’ conclusion is that, overall, agile methods could suit the needs of large organizations, in particular for small and collocated teams. However, integrating agile into the company environment could be challenging. Lagerberg et al. \[9\] report based on a survey at Ericsson that applying agile on a large scale facilitated knowledge sharing and effective coordination. In a systematic literature review on the adoption of agile methods at scale, Dikert et al. \[8\] identify 35 challenges, e.g., coordination in a multi-team environment with hierarchical management and organizational boundaries. In a position paper by Eklund et al. \[43\], research challenges of scaling agile in embedded organizations are presented. These challenges include, e.g., coordination of work between agile teams or taking into account existing ways of working for systems engineering. Similarly, Berger and Eklund \[10\] present, based on a survey with 46 participants, expected benefits and challenges of scaling agile in mechatronic organizations, including efficiently structuring the organization, understanding of agile along the value chain, and adaptation to frequent releases.

With respect to agile RE or RE in combination with the use of agile methods, there is less existing work. Based on a mapping study with 28 analyzed articles, Heikkilä et al. \[16\] conclude that the definition of agile RE is weak. Furthermore, they report several problematic areas such as the use of customer representatives, prioritization of requirements or growing technical debt. In a case study by the same authors at Ericsson, the flow of requirements in large-scale agile is studied \[23\]. Perceived benefits include increased flexibility, increased planning efficiency, and improved communication effectiveness. However, the authors also report problems such as overcommitment, organizing system-level work, and growing technical debt. Similarly, Bjarnason et al. \[54\] investigate the use of agile RE in a case study with nine practitioners at one large-scale company transitioning to agile. The authors report that agile methods can address some classical RE challenges, such as communication gaps, but cause new challenges, such as ensuring sufficient competence in cross-functional teams. In a case study with 16 US-based companies, Ramesh et al. \[19\] identify risks with the use of agile RE. These are, e.g., the neglect of non-functional requirements or customer inability. A systematic literature review on agile RE practices and challenges reports eight challenges posed by the use of agile RE \[15\], such as customer availability or minimal documentation. However, the authors also report 17 challenges from traditional RE that are overcome by the use of agile RE. The authors conclude that there is more empirical research needed on the topic of agile RE. Other studies have addressed the use of traditional RE practices and agile RE. Paetsch \[17\] provide a comparison between traditional RE approaches and agile software development while identifying possible ways in which agile software development can benefit from RE methods. The authors conclude that agile methods and RE are pursuing similar goals in key areas like stakeholder involvement. The major difference is the emphasis on the amount of documentation needed in an effective project. Meyer, in contrast, regards the relationship between RE and agile more critical, describing the
Figure 2.1: Overview of multiple case study research design

discouragement of upfront analysis and the focus on scenario based artifacts (i.e. user stories) as harmful [30], however not based on empirical data.

In summary, there is substantial existing work on the adoption of large-scale agile in system development, including empirical studies. However, existing work either focuses on identifying and evaluating agile RE practices [15, 16], or at presenting the current state of practice at single companies [23] and without explicitly targeting system development [54]. Hence, additional empirical work is needed to understand the complex phenomenon of agile methods and RE in the domain of large-scale system development. Our study contributes with a cross-case analysis of large-scale agile development and the role and scope of RE in this context. Specifically, we aim to compare how agile methods are adopted with different scopes in multiple companies (RQ1). Additionally, we aim to characterize the role of requirements (RQ2) and requirements-related challenges (RQ3) within this scope.

2.3 Research Methodology

In our multiple case study [76], we investigate one telecommunications company, two automotive companies, and one company developing software-intensive embedded systems (referred to as the Technology Company in this paper). All four cases represent large companies developing products and systems that include a significant amount of software, hardware, and (with exception of the Telecom Company) also mechanical components. All case companies have experience with agile software teams and have the goal to further speed up the development of their software intense systems.
Table 2.1: Data Sources

<table>
<thead>
<tr>
<th>Type</th>
<th>Company</th>
<th>Role(s)</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus Group</td>
<td>Telecom</td>
<td>2x Test Architect, System Manager</td>
<td>FG-1</td>
</tr>
<tr>
<td>Focus Group</td>
<td>Automotive 1</td>
<td>Process Manager, Specialist Platform Software</td>
<td>FG-2</td>
</tr>
<tr>
<td>Focus Group</td>
<td>Telecom</td>
<td>2x Test Architect, System Manager</td>
<td>FG-3</td>
</tr>
<tr>
<td>Focus Group</td>
<td>Automotive 2</td>
<td>System Responsible, 2x Function Owner, System Quality Engineer</td>
<td>FG-4</td>
</tr>
<tr>
<td>Focus Group</td>
<td>Technology</td>
<td>RE Change Agent, Chief Engineer</td>
<td>FG-5</td>
</tr>
<tr>
<td>WS</td>
<td>Automotive 1</td>
<td>Verification Manager, Specialist Platform Software</td>
<td>XComp 1</td>
</tr>
<tr>
<td>Telecom</td>
<td></td>
<td>Test Architect, System Manager</td>
<td></td>
</tr>
<tr>
<td>WS</td>
<td>Automotive 1</td>
<td>Verification Manager, Specialist Platform Software</td>
<td>XComp 2</td>
</tr>
<tr>
<td>Telecom</td>
<td></td>
<td>Test Architect, System Manager</td>
<td></td>
</tr>
<tr>
<td>Automotive 2</td>
<td></td>
<td>Test Architect, System Manager</td>
<td></td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td>Chief Engineer Software</td>
<td></td>
</tr>
<tr>
<td>Int</td>
<td>Telecom</td>
<td>Test Architect</td>
<td>T-TA-1</td>
</tr>
<tr>
<td>Int</td>
<td>Telecom</td>
<td>System Manager</td>
<td>T-SysM-1</td>
</tr>
<tr>
<td>Int</td>
<td>Telecom</td>
<td>System Manager</td>
<td>T-SysM-2</td>
</tr>
<tr>
<td>Int</td>
<td>Telecom</td>
<td>Developer</td>
<td>T-Dev-1</td>
</tr>
<tr>
<td>Int</td>
<td>Telecom</td>
<td>Scrum Master</td>
<td>T-ScrM-1</td>
</tr>
<tr>
<td>Int</td>
<td>Telecom</td>
<td>Area Product Owner</td>
<td>T-APO-1</td>
</tr>
<tr>
<td>Int</td>
<td>Telecom</td>
<td>Operational Product Owner</td>
<td>T-OPO-1</td>
</tr>
<tr>
<td>Int</td>
<td>Automotive 1</td>
<td>Technology Specialist (Safety)</td>
<td>A1-TS-1</td>
</tr>
<tr>
<td>Int</td>
<td>Automotive 2</td>
<td>Component Design Engineer</td>
<td>A2-CDE-1</td>
</tr>
<tr>
<td>Int</td>
<td>Automotive 2</td>
<td>System Design Engineer</td>
<td>A2-SDE-1</td>
</tr>
<tr>
<td>Int</td>
<td>Automotive 2</td>
<td>Function Owner</td>
<td>A2-FO-1</td>
</tr>
<tr>
<td>Int</td>
<td>Automotive 2</td>
<td>Function Owner</td>
<td>A2-FO-2</td>
</tr>
<tr>
<td>Int</td>
<td>Automotive 2</td>
<td>Software Developer</td>
<td>A2-SD-1</td>
</tr>
<tr>
<td>Int</td>
<td>Automotive 2</td>
<td>Software Developer</td>
<td>A2-SD-2</td>
</tr>
<tr>
<td>Int</td>
<td>Automotive 2</td>
<td>Product Owner</td>
<td>A2-PO-1</td>
</tr>
<tr>
<td>Int</td>
<td>Automotive 2</td>
<td>Scrum Master</td>
<td>A2-SM-1</td>
</tr>
<tr>
<td>Int</td>
<td>Automotive 2</td>
<td>System Tester</td>
<td>A2-ST-1</td>
</tr>
<tr>
<td>Int</td>
<td>Automotive 2</td>
<td>Functional Tester</td>
<td>A2-FT-1</td>
</tr>
<tr>
<td>Int</td>
<td>Automotive 2</td>
<td>Software Quality Expert</td>
<td>A2-SQE-1</td>
</tr>
<tr>
<td>Int</td>
<td>Technology</td>
<td>Requirements responsible</td>
<td>Tec-SRR-1</td>
</tr>
</tbody>
</table>
Sampling and Data Collection  Figure 2.1 gives an overview of our research design. Starting from a common case study design and common research questions, we conducted a cross-company scoping workshop (XComp 1 Scoping WS) to secure commitment from participating companies, align the goals of the study and finalize the research design. We then scheduled individual scoping workshops (Scoping WS) with each company, except for Technology Company, which, despite genuine interest in the study, could not free up resources for this study at that time. During these scoping workshops, we selected with help of our company contacts the most appropriate case in terms of availability and available experience on the topic, e.g., a specific product or component (partially) developed with the use of agile methods. These cases were selected to accommodate two aspects: variation to allow better generalization of results and convenience, since there was an interest to investigate the research questions in each particular case. This allowed us to cover a variety of perspectives during data collection, i.e., system overview, customer experience, development, integration, and testing. Our generic data collection instrument can be found online. Data collection was adjusted according to each individual case based on resource availability and commitment. For instance, the Telecom case relates to a large product development by many Scrum teams and we relied on a focus group followed by interviews (denoted Int in the figure) with a variety of roles (see Table 2.1). In contrast, the Automotive 1 Case relates to one Scrum team and we choose a focus group with the entire team, complemented with an interview of a safety expert. Interviews lasted approximately one hour and followed a similar interview instrument for all companies with domain specific adjustments for each company. For focus groups and cross-company workshops we scheduled three hours.

Not all researchers participated in all interviews and focus groups, but for each case we had one dedicated researcher who was present in all data collection events of that particular case. We recorded interviews and focus groups where possible and had at least two researchers take notes otherwise.

Data Analysis  For data analysis, we relied on a thematic coding approach. For each case, at least two researchers familiarized themselves with the data and highlighted noteworthy statements and assigned a label or code to each. Based on a card sorting approach, we then in the entire group of researchers discussed and iteratively combined codes into 30 candidate themes, from which we derived four high-level clusters containing 3-5 themes each. To validate the clusters, we discussed the outcome of our analysis in a reporting workshop with all participating companies.

Threats to Validity  By design, the external validity of case studies is low. Hence, generalization of our findings might not be possible to different companies or domains. In particular, we cannot reason about challenges for small-scale or pure software development. We believe that while some challenges might be visible there as well, they can likely be managed ad hoc or within the scope of agile practices. We designed our study to identify common challenges across participating companies. Thus, our research method does not support

---

1http://grischaliebel.de/data/research/KGLKK_re_agile.pdf
2We refer to participants of interviews and focus groups as interviewees
any deep argument about differences between companies, domains, and market positions. However, given that we found similar themes in all four cases, we expect that these apply similarly to other companies or projects in large-scale systems engineering.

To increase internal validity, we used data triangulation between interviews and between case companies. Further, the results of our analysis were discussed in a cross-company workshop (XComp Validation WS) with the companies. The workshop included key roles from each company that were already involved in the study. We also used the workshop to discuss underlying root causes and challenges that are shared by all companies (see Section 2.5). To avoid a too restricted view on smaller parts of a project or a product, we selected interviewees from different parts of the development, including at least one team and several system level roles in each case. We relied on a convenience sample and companies provided us with access to dedicated experts in the research field (agile transformation, RE) with a genuine but diverse interest in the field. While we hope that this improved internal validity, it might have introduced a selection bias, which we tried to mitigate by encouraging participation of both proponents and opponents of agile/RE.

We limited threats to conclusion validity by improving the interview instruments in multiple iterations and by conducting interviews in pairs of two researchers. With all four case companies, we have a prolonged involvement and therefore a mutual trust among the parties exists. The data analysis was discussed and refined among the authors in several iterations.

2.4 Findings

This section presents our findings in relation to our research questions. Fig. 2.2 presents an overview of our themes.
2.4.1 What are possible scopes of applying agile methods in large-scale system development? (RQ 1)

2.4.1.1 Context of Case Companies

Figure 2.3 gives an overview of different scopes we identified in our cases. The next Section 2.4.1.2 elaborates on common themes across these scopes.

**Telecom Company** The Telecom Case relates to the development of one major product. More than 30 Scrum teams develop in parallel based on a scaled agile approach (adopted from SAFe [82]). Scrum sprints are based on a backlog and a hierarchy of product owners breaks down product requirements and customer visible features to backlog items. While these product owners represent the customer requirements towards the product development, system managers represent a system requirements perspective. The overall effect is a continuous development stream and feature flow, which is supported by a powerful infrastructure that enables continuous integration and testing. Pre-development generates knowledge about new features, which enables effective planning for continuous delivery.

Particular to the Telecom case, hardware development is largely decoupled from the software development. New hardware becomes available with a regular, but low frequency. Thus, the software development sets the pace of system development, which can be seen as continuous and agile, in that it embraces agile values as much as possible. In Figure 2.3 this is shown by the largest, dotted box, which implies that the whole scope of a traditional V model is covered.

**Automotive Company 1** In Automotive 1, agile methods have been successfully applied to in-house development of software components. In the light of growing competition from software-centric companies, e.g., on autonomous driving, there is a desire to scale up these fast-paced approaches from developing software components to developing complete functions, thus including agile development of hardware and mechatronic. The selected case is a pilot project that re-implements a whole customer function in an agile way. Integration of
2.4. FINDINGS

this function into a real vehicle requires additional verification with respect to safety and overall system behavior. Thus, we would characterize this situation with the second largest box in Figure 2.3, where a function owner takes responsibility for one particular function and implements it with an agile team.

Automotive Company 2 With Automotive 2, we selected a case responsible for safety critical functionality developed in house. As with Automotive 1, agile teams develop software. For this, software requirements are transformed into backlog items. In order to speed up development of this differentiating functionality, different measures have been taken to speed up the overall system development. While this development still corresponds to the V model, the introduction of a shared information model that supports storing requirements, design elements, tests, and implementation models helps to shorten the development time significantly. In FG-4, participants referred to this approach as narrow V model (comparable to agile loop in [43]). In Figure 2.3, we describe this as the smallest box, not to refer to overall development speed, but to the fact that hand-over between plan-driven and agile development happens on a low level of abstraction.

Technology Company The Technology Company develops mechanical products, both for consumer markets and for industrial development and manufacturing. Their system development includes several system elements. Software development is mostly confined to two of these elements, both of which are characterized by agile methods and practices such as Scrum and Continuous Integration. As with Automotive 2, we refer to this situation with the smallest box in Figure 2.3.

2.4.1.2 Agile Scope in Large-Scale System Development

Summarizing the four cases, we recognize that some case companies have come a long way towards continuous software engineering and enterprise-wide adoption of agile [91]. Others are currently moving in that direction. Our research aims for common themes, regardless of the scope of agile adoption (for which we control with RQ1) or agile maturity (which we did not explicitly investigate in this study). In the analysis of interview data, we uncovered the following themes that relate to the scope of agile methods.

Agile Islands in a Waterfall From a product perspective, a plan-driven or stage-gate approach is important. Release of a new product needs to be planned and longer development cycles for hardware and mechanical components need to be scheduled. All of our case companies have agile software development teams that operate within the context of a larger system engineering process, which one interviewee described as agile islands:

“It feels like agile islands in a waterfall.” — FG 2

The challenge we found here regardless of agile scope in the specific case is continuous information exchange between plan-driven and agile parts of an organization. Incubation of new innovative ideas, facilitating quick feedback loops, and quick learning on potential business value are important assets to
remain competitive, yet they are hard to integrate into the overall system development approach in all our cases.

**Component vs. Systems Thinking** The scope of agile development also relates to the relationship between components and the system that is built on it.

“We need to balance project autonomy, freedom, empowerment with platform reuse, consolidation of data, and unambiguous requirements. The teams have a lot of tacit knowledge, which is not available beyond their scope. But how much ceremony should we force on teams?” — FG 3

Even beyond designing a single system, knowledge about customers and their needs should be maintained for future projects. Without a good knowledge management approach, this can collide with the desire to allow empowered component teams to make fast, local decisions.

**Safety Critical and Agile** Most of our case companies are subject to safety standards and regulations. Several participants in FG2 and FG4 expressed concerns that the development of safety critical software together with corresponding standards could impede agile development. As examples, the participants expressed the need for documentation and tracing that is required by several standards, such as ISO26262. However, an expert for functional safety in Automotive 1 stated that the need for documentation and tracing is related more to the size of the company and the system rather than regulations.

“Many see that as a problem. Many say that it’s a safety problem, it is a 26262 problem. But we say [... we need to document anyway since then half a year later it is a different team [working on the same software]” — A1-TS-1

According to our interviewees, standard conformance could be combined with agile development if only this was planned in a systematic fashion, e.g., by sandboxing safety critical parts.

Still, the need to guarantee safety relates to both component and system level. Depending on the scope of agile methods, this introduces interfaces between plan-driven parts of the organization and agile teams.

**Impact of new features on infrastructure** In system development, integration testing often depends on a strong laboratory setup that allows testing hardware, software, and potentially mechanics together. While a new feature might mainly depend on changes of software and can be provided in an incremental, fast-paced way, it could require an update of the test environment, which may include sophisticated hardware and environment models. However, changing the test environment might take as long as finishing the software components, thus introducing delays, if not started in due time. Similar concerns relate to other infrastructure for continuous integration, delivery, and deployment.

This theme shows that independent of the scope in Figure 2.3 there is a need to maintain a system-level perspective beyond self-organized teams and to allow requirements related information to escalate to this level as early as possible.
2.4. FINDINGS

Time for invention and planning Study participants in Automotive 1 reported that an exploration of solution space is difficult within agile sprints, as it would be impossible to commit to a fixed schedule without deep knowledge about new features. Pre-development is required to better understand the impact of new features. If this is done by a dedicated group, this would imply documentation and hand-over of results and slow down the process. As a remedy, specific exploration sprints were brought up. Another solution could be to transfer engineers between pre-development and agile system development, so that they can also share their knowledge with team members. Either way, this highlights the need for good scoping of agility within system development.

To answer RQ1, the four cases show that agile development is indeed applied on very different organizational levels. Irrespective of this scope, common issues with RE occur. Handovers between the boundary of agile and plan-driven parts of the company even occur in the Telecom case.

2.4.2 How is the role of requirements characterized in large-scale agile system development? (RQ 2)

Requirements: Order, Goal, or Dialogue? In all our cases, software development teams are enthusiastic about agile methods. As a consequence however, many see requirements as something that comes into the way of being agile. A requirement is seen as an order, interfering with the empowerment of teams. Instead, developers tend to prefer user stories, that set goals, but do not limit the agile team’s autonomy in deciding how to reach them, i.e., to claim ownership.

According to our interviewees, agile methods imply a bidirectional flow of information, since they advocate to explore the best way of satisfying a customer need through incremental and iterative work. Agile teams are used to uncover new aspects, details, or even new requirements. With respect to RE, the main advantages brought up by interviewees on all levels are related to ownership and include consequent orientation towards user and customer value as well as the fact that developers have the mandate and competence to update user stories within the scope of agile development. Thus, user stories and implementation are usually extremely consistent. This fast paced agile learning however needs to be fed back to system-level requirements models, a task that is currently challenging in all our cases.

Embrace Change of Requirements? Sufficient facilities for updating system requirements based on agile learning are currently missing. Thus, such updates are manual work, leading to inconsistencies, which are expensive to remove and can be considered waste in the overall development process. In addition, developers have little intrinsic motivation to update requirements models based on updates to user stories, as they are not part of their delivery (usually code and tests). If however requirements updates were not propagated, the system requirements view would become quickly obsolete and detached from the real system. Consequently, roles responsible for customer and high-level system requirements (product owners, function owners, system managers), fear a loss of important knowledge for later maintenance of the systems. A more
systematic approach to manage requirements updates received from agile teams would make their jobs much easier.

Another key challenge highlighted by our interviewees relates to the timeliness of required information. Agile development is for example able to digest changes and information that only incrementally becomes available, but it struggles with information provided in bulk.

**Requirements as Technical Documentation?** With respect to a wider adoption of agile methods and their rejection of requirements, we recognized across all companies a tendency to encounter more inconsistencies between requirements and implementation. This is partly due to the aforementioned lack of propagating changes of user stories to the system requirements model. While this relates to classic requirements challenges, the effect of introducing agile methods has shifted the problem somewhat: agility helps investigating assumptions as well as resolving ambiguities and inconsistencies early. Knowledge gained in this way has high value as a reference for maintaining the system, i.e., as technical documentation. But as long as this knowledge is not efficiently shared in the whole development organization, the challenge remains effective.

For technical documentation, some developers would prefer to use automated regression tests, since this would put less overhead on their work. Most of our interviewees however, especially on higher levels of abstraction in system development, indicated that higher-level requirements are of critical importance here, since reconstructing the intended behaviour of a component from a set of a dozen test cases and hundreds of user stories renders them useless for technical documentation. We refer to the challenge of tests and user stories being insufficient for specification in the next section.

### 2.4.3 Which requirements related challenges exist in large-scale agile system development? (RQ 3)

With respect to RQ3, we see two communication related areas of challenges: *Shared Understanding of Value* and *Build and Maintain System Understanding* (see Fig. 2.2).

#### 2.4.3.1 Shared Understanding of Value

**Customer value to team** Agile development has been adopted for all our case companies as presented in Section 2.4.1.1. Study participants indicate a large distance between customers and developers:

“We have so many different levels between the real customer, the customer units and [the teams], with each transaction you lose so much content” — T-TA-1

In all our cases we found dedicated roles that channel information from multiple stakeholders down to the teams. It is not trivial to bridge that gap, direct interaction of teams and stakeholders can lead to chaos when established plans are circumvented, on-site customers are not an option, and product-owners hard to implement.

However, one participant exclaimed that the focus on agile practices occupied the teams so much that this caused a neglect of product value.
2.4. FINDINGS

“The idea of producing value for the customer has been discarded. Teams just want to be agile (i.e., independent)” — XComp 1

However, value creation is not solely the teams’ responsibility as the requirements breakdown starts from the customer units, in the Telecom case, or from the function management units, in the Automotive cases.

To ensure frequent delivery, application of agile methods implies breaking down large features into smaller sub-features and tasks that can be finished during a typical Sprint. This is the practice in all our case companies even though the used methods differ.

“[...] usually, a feature is what you can sell to customers but teams talk of sub-features [...]” — T-SrM-1

One interviewee (T-APO-1) pointed out that it is hard to break down the requirements such that they carry user value, a challenge also recognized in other cases (FG-2 and FG-5).

**How to write meaningful user stories** User stories provide a fast means to share knowledge both on a high and a low level in an agile system development. In the Telecom as well as in the two Automotive cases, user stories are used for two purposes as one interviewee says

“... so there are user stories that of course take the view from the end customer and describe what the end customer wants from our system and why. But then there are other user stories that are more like work descriptions of what the team should achieve and those could be like internal things that need to be developed in order to keep the architecture constrained.” — T-SysM-2

A Function Owner in FG-4 (Automotive 2) specifically expressed that high-level user stories could help to communicate value early. However, as discussed in Theme a) *customer value to team*, it is particularly difficult to write user stories that have direct value for the user. Such user stories would typically be too large to be completed and demonstrated in one sprint. Yet, breaking it into more user stories could deteriorate requirements quality since not enough effort goes into maintaining the requirements. It is therefore hard to understand which user story can be traced to requirements. In summary, user stories are hard to write at the scale and complexity of the cases in our study, yet they offer a unique opportunity to bridge distances between customer and developer.

**Feedback and requirements clarification** In several companies, study participants raised the issue of long or complicated feedback cycles. At Automotive 1, one study participant named *slow mechanical or hardware development* as one of the main reasons for long feedback cycles. If software has to be tested together with actual hardware, feedback on software functionality is postponed until the hardware is ready.

One study participant stated a second reason — often *customers are not agile* and take a long time to try out and approve new features. By the time feedback then reaches the agile teams, they are already working on another part of the product and do not remember exactly what the feedback is about. That is, for the teams the feedback comes too late, while customers do not see value in giving quick and frequent feedback, even on smaller increments.
This challenge is especially encountered if the system under development is supposed to be integrated into a larger system at the customer site, as for example in the Telecom and Technology cases.

A third reason for complicated feedback cycles is that there is a large number of stakeholders, both external and internal. Due to the complex nature and the scale of the products developed by our case companies, there is rarely a single customer. Instead, requirements inflow occurs from many different sources, e.g., customers, authorities, managing subcontractors and sourcing, or standardization organizations. In many cases, requirements need to be discussed with and communicated to other stakeholders within or outside the organization, delaying feedback.

Even if feedback and requirements clarification can be facilitated despite the challenges raised here, gained knowledge must be effectively managed as participants in FG-5 (Technology Case) pointed out. Two aspects of this challenge were raised: First, it is unclear where knowledge about a specific customer can be managed beyond the team and current project. Secondly, in continuous product development, teams might not realize that they have valuable knowledge for other parts of the system development, while those other parts do not know that valuable knowledge is available.

2.4.3.2 Build and Maintain System Understanding

Inform and Synchronize Between Teams Teams receive requirements from the product managers through several organizational levels in the Telecom case. Furthermore, they often need to exchange information with other teams to synchronize the development. This process of channeling the ‘right’ information towards and between teams is difficult and time-consuming. Hence, it limits agility and speed of teams.

A similar issue was raised at Automotive 1, where participants discussed which organizational scope agile methods should have with respect to time for planning and invention (Section 2.4.1.2). FG-2 participants wondered whether agile should be limited to the development only, or should start from a feature request. In the former case, developers would receive feature requests in the form of already broken down requirements for implementation. In the latter case, developers would have to do the breakdown of a feature request into smaller units themselves. While both cases seem to be feasible, the question is how teams can be synchronized in any of these cases. If requirements are broken down by an external role or team, possibly in a plan-driven way, they need to be handed to different agile teams and their work needs to be synchronized. If they are broken down and implemented within one team, multiple agile teams only need to synchronize when there is interaction with or dependencies to features developed by other teams. Awareness about such dependencies is a pre-requisite.

Creating and Maintaining Traces Several interviewees in the Telecom case stated that their system requirements work as a documentation of what the system is doing, rather than a plan of what shall be implemented. In their view, system requirements are used as an input for specifying test cases and to allow analysis for new features, not as a plan for implementation.
“You can’t really afford to have this kind of static requirements work upfront which will be a waste anyway when you implement stuff. The way we handle requirements now is more like a system description.” — T-TA-1

Since user stories relate directly to feature implementation and are not always systematically derived from existing requirements, direct tracing is not always possible. A similar situation occurs in Automotive 2, where product owners write user stories based on plan-driven requirements they receive as an input. These user stories can in fact be rather local development tasks and backlog items that do not require tracing to system requirements. Thus, traces are not systematically managed, which can lead to additional work in cases where such backlog items become relevant for tracing to system requirements. The fact that often only the product owner is aware of which user stories originate from which requirements can slow down collaboration between agile teams and plan-driven RE teams. Interviewees in the agile teams considered tracing user stories to requirements to be documentation, which should not be part of the agile process. Instead they preferred to spend their time on implementation:

“I don’t think traceability is not required or something like that. It’s just that my focus hasn’t been on documenting the function. I just focus on doing implementation and developing the function.” — A2-PO-1

This view was also shared in Automotive 1: While participants stated that tracing is valuable, or even required by standards, they felt that right now there is not enough incentive for agile developers to create traces. They wished for an incentive or directly visible benefit for the developers as well as for simplifying trace creation.

Bridging the Gap Between Plan-Driven and Agile Development In the Telecom case, we found that system managers feel disconnected from the agile teams. Their role is to be experts on a certain part of the system and support teams with their knowledge of the system requirements. However, as one interviewee stated they currently cannot be in contact with all teams and might therefore not get a notification if something has been changed with respect to existing requirements.

“If [...] a team updates a past requirement, perhaps I should get like a notification on that so I can ask them ‘Have you forgotten X?’.” — T-SysM-1

Similar challenges exist with the other companies, e.g. in the Automotive 2 case where agile teams can add new backlog items or change existing ones in collaboration with the product owner. However, since agile teams do not interact directly with system requirements (see b creating and maintaining traces), they do not consider knowledge about them to be of importance. Further, backlog items are easy to understand, even for stakeholders not directly involved, and allow them to share their opinion. While this is generally perceived positively by the interviewees, it was also brought up that this can cause the function owner to be overexposed to change requests. One function owner expressed this as follows.

“The more people look into requirements, the more they read them, the more iterations it will become. [...] there is going to be more opinions, comments and also more work.” — A2-FO-1
As this can lead to inconsistencies between changed and new backlog items and the system requirements, e.g. in the case where a system requirement related to a new user story already existed, increased gatekeeping becomes necessary. This generates effort for backlog grooming by the (agile) product owner, and managing of system requirements by the (plan-driven) function owner. The current separation between both worlds does not seem to be ideal, since product and function owner can easily become bottlenecks, and late resolution of inconsistencies can create additional effort. If the actual implementation deviates from the original requirement or when some requirements are not implemented, this will surface as problems during system integration and testing. Tests are developed against the plan-driven requirements and are therefore in need of an up-to-date version.

“If I have a requirement saying this thing should happen, when I test it, I find out that what is supposed to happen doesn’t happen. [...] And then I find out the requirement wasn’t updated. So actually the implementation was correct but the requirement isn’t matching the implementation.” — A2-ST-1

Further, if the system has to be evolved or maintained in the future, outdated requirements can cause misunderstandings.

**Tests and User Stories not sufficient** The idea of using test cases both as actual test artifacts and as requirements documentation is wide-spread in the agile community [30] and was also discussed by several participants. While this was seen as a potential way to reduce documentation effort, several issues with this approach were brought up. According to several study participants, user stories and test cases do not carry enough information to serve as a means of documentation:

> “Tests are written in a pragmatic way. They do not capture the 'why'.” — Tec-SRR-1

Other interviewees throughout the companies added that one would need a number of tests to document any significant requirement, which will then be hard to reconstruct from just reading the tests during maintenance.

Several interviewees saw similar problems with user stories, as they would only reflect single scenarios. The overall system behavior would then emerge from the syntheses of all these single scenarios. To derive this full picture from tests or user stories only would, however, be too difficult:

> “If we don’t specify this kind of complete [requirements] specification, we could try to use all [...] user stories [...] But then we must base the understanding on [...] lets say [...] 2000 user stories [...] and try to find a good way of describing the complete system.” — T-SysM-2

It is interesting to note that this challenge surfaces early on, i.e., when an incoming customer request is analyzed. Therefore, if agile teams only develop backlog items based on finished requirements that they receive from other parts of the organization, they might not be aware of this challenge and therefore consider test cases to be sufficient.

While in the Telecom case the issue of understanding system behavior from user stories or tests was mainly discussed with respect to new features, participants in Automotive 1 raised it especially for system maintenance. FG-2
participants agreed that user stories or test cases would not be appropriate to understand the behavior. They were unsure what form of documentation should be used instead, which level of detail the requirements should be on, and how they could be different from ‘traditional’ requirements.

**Establishing an agile tool chain** Several participants in FG-2 (in Automotive 1) raised the topic of an adequate tool chain for agile RE. They reported that, at the moment, they use traditional tools for requirements management and tools that are aimed at agile development, such as JIRA, for the actual development. These tools are however largely separated and, thus, they felt that in order to be able to perform RE in a more agile way, they would need appropriate tools. This is an interesting issue, as a simple issue tracking tool is not likely to address their needs as they operate in a multi-disciplinary, regulated environment. In particular, there are formal requirements for traceability imposed by safety standards such as ISO26262.

Similarly, in the Telecom Case, current tooling was brought up as a hindrance for speed and agility. Interviewees described the current process of updating system requirements as too slow and cumbersome. They stated that by introducing a more efficient tool solution, engineers could potentially be more motivated to make changes to requirements and by this narrow the gap between agile user stories and requirements.

The need of a tool-chain that better supports agile information flows was confirmed by the other two cases as well.

### 2.5 Discussion and Implications

Even though the four cases differ in their context, i.e., domain and scope of agile methods within system development, we found common concerns and challenges with respect to RE. Based on our findings, we draw the following four conclusions:

**Conclusion 1:** Challenges of RE for large-scale agile system development relate to communication and knowledge management. While related work implies that communication challenges are mitigated by agile approaches and less prominent in agile RE [15,54], all our challenges relate to communication and knowledge management. Both aspects are at the core of Agile and RE, indicating a need for fundamental research in these areas specifically for system development.

**Conclusion 2:** Challenges relate to two areas of requirements knowledge: User Value and System Understanding. We identified two categories of RE challenges in the large-scale agile system development domain; shared understanding of value and building and maintaining system understanding. While pre-agile RE approaches differ between user and system requirements specifications, we are not aware of related work that makes this distinction for RE in the scope of agile development. Surprisingly, we found that companies were not very interested in agile RE practices themselves (which is the focus of the majority of related work, as [15] indicates). In contrast, they found it more important to understand how RE can support agile methods in large-scale system development and how agile development can be integrated with existing
processes. Our findings suggest that such support cannot be offered sufficiently by traditional, upfront RE, as indicated \cite{23,30}. Similarly, we did not find any specific roles that emerge in the large-scale agile environment, comparable to the roles presented in \cite{92}. Our results suggest that continuous and agile development methods on a large scale require new concepts.

**Conclusion 3:** Challenges relate to the interplay of stakeholders from three domains: customer, development, and integration & testing. The development domain is generally embracing agility and characterized by a dislike for traditional requirements and bulk updates. The require better **synchronization between teams** and wish for **establishing an agile tool-chain**. In contrast, the customer domain is concerned with breaking down customer-visible features in order to communicate **customer-value to team**. They require better support for **writing meaningful user stories** and for **bridging the gap between plan-driven and agile development**. The integration and testing domain is struggling to **create and maintain traces** and with the fact that **user stories and tests** are not sufficient to build and maintain sufficient system understanding.

**Conclusion 4:** In order to yield their full benefits, agile practices and a **holistic system requirements model must be better aligned.** Key challenges occur when there is an interaction, or a lack thereof, between the three domains and we believe that industry would benefit from new impulses from research in the following areas.

**Bridge distance between customer and developer** We found in all four case companies that the distance between the customers and the development is perceived to be too large. In particular, it was described as difficult to break down a feature request into small packages that both have customer value and can be delivered in small iterations. However, agile values such as individuals and interactions \cite{93} as well as agile practices such as continuous delivery \cite{94} depend on a good notion of value. Yet, we found this particularly hard to establish in large-scale system development, because of unclear customer role and scale. In case of an external customer, any customer visible feature will imply more work than can be done within one sprint or by one team. This makes feature decomposition necessary and it is impossible for a single team to demonstrate customer value at the end of a typical sprint. Related work in this direction has in particular pointed out challenges with the practice of customer representatives \cite{15,16,19}, but it seems that the notion of value itself might be problematic.

**Establish information flow and knowledge management** Similarly, the information flow between the domains currently does not well support agility, in the sense that information exchange and feedback cycles are often too slow. As our results suggest, it is crucial to establish suitable exchange and management of knowledge throughout large-scale agile system development. Agile development works best with a continuous inflow of new requirements and can in turn help to resolve ambiguities and refine requirements just in time, as new knowledge becomes available. However, it is important to support updating system requirements models and to coordinate the information flow between parallel teams. This finding suggests that communication issues continue to
be relevant in large-scale agile RE, in contrast to what is suggested by related studies, e.g., [15, 54].

**Support analysis of incoming requirements** In all case companies, the need for documented requirements was stated because of a need to understand the current system in order to analyse new feature requests or to maintain the system. Also, several interviewees stated explicitly that using test cases as requirements would not be enough to fulfill this need. Therefore, we see the need for more work investigating the use of different notations, techniques or methods to inform early analysis of incoming requirements. While the user story format has been described as insufficient [16], test cases are often named as an alternative [16, 95]. Our findings suggest that using test cases, even in combination with user stories might not be sufficient, in particular with respect to supporting the understanding of a system’s current functionality.

**Dealing with system qualities and regulations** As our results suggest, companies are facing challenges when trying to address quality requirements, such as safety and security. Such quality requirements often relate to regulations that can directly challenge agile practices, by requesting end-to-end documentation and tracing of safety-related requirements. While these regulations might be one argument to have requirements documented, interviewees in all companies felt a strong need for documentation even without regulations. Especially the need to understand the current system in order to react to new feature requests or system maintenance were raised in all companies. Traditionally, long upfront analysis and planning aimed to address these needs [30]. However, as companies try to speed up their development, research needs to investigate new ways of dealing with documentation of such cross-cutting issues. Ensuring qualities and addressing non-functional requirements has been brought forward as a challenge in agile RE [15, 16], and first works exist to address regulations in agile [67].

In summary, companies need to aim for an open dialogue to balance system development needs with allowing agility. To address requirements updates and regulative issues, one could include explicit updates to requirements as part of each sprint, e.g. to accommodate safety regulations [67, 68]. To address the lack of intrinsic motivation to update requirements or establish tracing, immediate feedback loops should be established that show value to developers [96]. In addition, gamification approaches resonated well in discussions with practitioners. To address the distance to customer and the challenge of none-agile information flow to development, one could investigate how to best use user stories on a high level of abstraction and early on as well as to facilitate a dialogue between roles in different domains [93].

## 2.6 Conclusion and Outlook

We presented our results from a multiple-case study with four systems engineering companies on the interaction of RE and agile methods in large-scale development. We studied the scope of agile methods, the role of requirements in this context, and requirements-related challenges of large-scale agile systems
development. In all case companies, the way how plan-driven and agile development currently co-exist within the systems engineering environment limits the potential development speed. We found that in all companies, there is a need for some kind of traditional RE, especially with respect to documenting a system’s behavior for future feature requests or maintenance. The case companies use different scopes for agile development, ranging from agile development on team-level embedded in an overall plan-driven process up to agile development for the entire product development. Despite the different scopes, we observed similar challenges in all companies. These relate to establishing a shared view of value from the customer and other stakeholders down to development, and to building up and maintaining a shared system understanding. In order to mitigate these challenges, we encourage future work to focus on aligning a holistic requirements model with agile practices. Ideally, this will allow large-scale system development efforts to fully benefit from agile methods, while still systematically managing knowledge about customer value and how the system under construction relates to this.
Chapter 3

Paper B

Safety-Critical Systems and Agile Development: A Mapping Study

R. Kasauli, E. Knauss, B. Kanagwa, A. Nilsson, G. Calikli

Proceedings of 44th Euromicro conference on Software Engineering and Applied Applications (SEAA’18), 2018
Abstract

In the last decades, agile methods had a huge impact on how software is developed. In many cases, this has led to significant benefits, such as quality and speed of software deliveries to customers. However, safety-critical systems have widely been dismissed from benefiting from agile methods. Products that include safety critical aspects are therefore faced with a situation in which the development of safety-critical parts can significantly limit the potential speed-up through agile methods, for the full product, but also in the non-safety critical parts. For such products, the ability to develop safety-critical software in an agile way will generate a competitive advantage. In order to enable future research in this important area, we present in this paper a mapping of the current state of practice based on a mixed method approach. Starting from a workshop with experts from six large Swedish product development companies we develop a lens for our analysis. We then present a systematic mapping study on safety-critical systems and agile development through this lens in order to map potential benefits, challenges, and solution candidates for guiding future research.
3.1 Introduction

A system is safety-critical if its failure can cause financial loss, damage to the environment, injury to people and in some cases, loss of lives \cite{97,98}. The use of software in safety-critical systems has continuously increased to an extent where software failures can impair system safety. Examples in the automotive domain include software in a break system, which in case of failure could result in unacceptable hazards, but also active safety functions that override driver behavior in certain situations to avoid a crash. Similar examples can be found in other domains (e.g. railway \cite{73}, avionics \cite{71}).

Safety-critical systems (SCS) are heavily regulated and require certification against industry standards by the relevant governing body \cite{99}. For that reason, they have to be designed with safety in mind. However, as opposed to the hardware where general safety design principles have been incorporated into standards, the standards for development of safe software are still developing. These standards, for instance ISO26262 for automotive, IEC61513 for nuclear, IEC62304 for medical and DO-178B for avionics aim to ensure that best engineering practice is followed and are often based on traditional, plan-driven approaches \cite{100}. This is related to the common criticism that agile methods neglect upfront analysis with a focus on scenarios for describing requirements \cite{30}. Because the standards are written like that and because people think that agile methods are not suitable, SCS development is most often characterized by highly-structured, plan-driven approaches such as V-model or waterfall \cite{101}.

While agile methods have significantly changed the way most software is developed, safety-critical software was widely excluded from this trend \cite{30}. Arguments for this exclusion relate to the need for documenting safety cases and the need to analyze and specify safety requirements upfront, both thought to be in conflict with agile values. Thus, companies have different approaches to develop software, depending on whether it is safety-critical or not (e.g. allowing faster time to market of non-critical software such as infotainment) \cite{30}. Recently, however, the situation has changed through potentially disruptive trends that significantly increase the need for short development cycles and quick time to market. Complex dynamic systems-of-systems require components of different vendors to interact at run-time and the dynamic nature of such environments can make continuous deployment a necessity for product success and for maintaining functional safety, e.g. in the automotive domain \cite{102}. The huge attention given to autonomous driving and intelligent vehicles has lead to a jump in complexity of SCS. Companies like Tesla have demonstrated how the ability to deploy new functions and explore their performance in the field can yield huge advantages. Software companies such as Google, Apple, and Amazon have pushed in the automotive market, increasing the need to develop competencies in continuous software engineering. Slowly, companies developing SCS realize the competitive advantages that agility can provide \cite{104}.

Existing reviews on agile development and safety \cite{103,104} mostly focus on determining whether agile methods can improve embedded systems development \cite{103}, in general and on how these agile methods are being adopted in these environments. Kaisti et al.’s study \cite{103} does not particularly focus on safety-critical systems whereas Cawley et al. \cite{104}, with a focus on regulated environments, found few works indicating a low but growing adoption of agile
3.2. METHODOLOGY

In order to enable future research in this important area, we present in this paper a mapping of the current state of practice based on a mixed method approach. Starting from a workshop with experts from six large Swedish product development companies, we develop a lens for our analysis. The aim of this mapping study is to evaluate and present the combination of agile development methods and SCS. Our goal is to provide a map that can enable future research and method development for researchers and practitioners. Specifically, we map suggested benefits, challenges, and solution candidates as well as their relation to up-front analysis, just-in-time activities of agile teams, and infrastructure to support these. We aim to answer the following research questions.

**RQ1:** What research exists about agile development of SCS?

**RQ2:** What are the key benefits of applying agile methods and practices in SCS development?

**RQ3:** What challenges exist with agile development of SCS?

**RQ4:** What solution candidates (e.g. principles and practices) promise to address challenges with respect to agile development of SCS?

Thus, this paper contributes an overview of existing research, potential benefits, challenges, and solution candidates of agile development of SCS. We discuss our findings in three dimensions: The necessity of up-front analysis, the ability to shift effort into just-in-time activities, and the potential of better infrastructure to support these. We expect this contribution to be of value for practitioners aiming to optimize their processes as well as for researchers who may base future research on our synthesis.

### 3.2 Methodology

In order to provide the required context while answering our research questions, two complimentary activities have been used; a cross-company workshop and a mapping study. This section provides an elaboration on these two processes.

**The Cross-Company Workshop:** We started to get interested in the interplay of agile methods and SCS based on ongoing collaborations and research projects on continuous integration and deployment. Through this existing network, we invited experts responsible for adoption of continuous integration, continuous deployment, and agile methods as well as safety experts from large Swedish companies to a workshop on SCS in continuous software engineering. During this workshop, representatives from six companies attended. We presented a very preliminary screening of literature, followed by presentations from industry participants. We then had an open discussion through which we arranged the information presented and agreed on critical aspects to be further investigated. From this activity, we derived a lens for conducting our mapping study to guide our research. We use this lens to provide an overview of research in the field, specifically focusing on identifying proposed benefits, challenges, and potential solutions and to discuss these findings based on their relevance to up-front analysis (i.e. activities to be done before agile teams can continuously add value through agile development methods), just-in-time...


<table>
<thead>
<tr>
<th>Inclusion/exclusion criterion</th>
<th>Number of documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Excluded</td>
<td>Included</td>
</tr>
<tr>
<td>Search from 2001 to 2017</td>
<td>1986</td>
</tr>
<tr>
<td>Limit results to subject <em>Computer Science</em> in Scopus</td>
<td>−1420 546</td>
</tr>
<tr>
<td>Exclude documents without authors</td>
<td>−81 465</td>
</tr>
<tr>
<td>Exclude non-english documents</td>
<td>−6 458</td>
</tr>
<tr>
<td>Exclusion based on title and abstract</td>
<td>−391 69</td>
</tr>
<tr>
<td>- Document mentions safety but not in the agile development context.</td>
<td></td>
</tr>
<tr>
<td>- Document discusses agile without a particular focus on SCS.</td>
<td></td>
</tr>
<tr>
<td>- The subject area was not software engineering or development.</td>
<td></td>
</tr>
<tr>
<td>- The document is not peer reviewed.</td>
<td></td>
</tr>
<tr>
<td>Criteria based on full text</td>
<td>−35 33 + 1</td>
</tr>
<tr>
<td>- Exclude documents where full text could not be obtained.</td>
<td></td>
</tr>
<tr>
<td>- Exclude documents to which an extended version was found and could be included.</td>
<td></td>
</tr>
<tr>
<td>- Exclude documents that do not allow extraction of context information.</td>
<td></td>
</tr>
<tr>
<td>- Include documents presenting an experience report.</td>
<td></td>
</tr>
<tr>
<td>- Include documents presenting a technical solution paper or an experiment.</td>
<td></td>
</tr>
<tr>
<td>- Include documents presenting a case study with students or practitioners.</td>
<td></td>
</tr>
<tr>
<td>- Include other documents (e.g. short papers) that include industry experience.</td>
<td></td>
</tr>
<tr>
<td>- Include papers obtained by snowballing given they meet the above criteria.</td>
<td></td>
</tr>
</tbody>
</table>

activities (i.e. activities done during agile development sprints), and long lasting infrastructure beyond a single project.

The mapping study: The key steps in the mapping study followed guidelines by Petersen et al. [80] while also borrowing some concepts from the established method of systematic literature reviews [81]. Kitchenham and Charters [105] outlined three phases of performing systematic reviews, i.e., planning, conducting and reporting the review. These phases were later simplified by Pertosny et al. [80] into five stages that include: defining research questions (see Section 1), conducting a search for primary studies, screening primary studies based on inclusion/exclusion criteria, classifying the primary studies, and data extraction and aggregation. These are explained in the subsections that follow.

3.2.1 Search strategy

Basing on our research questions, we selected the major search terms; "safety-critical systems" and "agile development". We then adapted the PICO (Population, Intervention, Comparison and Outcome) criteria proposed by Kitchenham and Charters [105] to construct the search terms used in our final search. With respect to *population*, we include terms related to safety-critical systems. These are synonyms or terms related to the major term and include safety-critical software, safety, safety critical, regulated, regulation or software intensive. With respect to intervention, we included terms relating to agile development and agile methods. These include agile, agility, continuous delivery, continuous
3.2. METHODOLOGY

deployment, continuous integration, scrum, xp, agile method, agile process and agile practices. Specifically, we relied on a literature review on agile methods \[106\] to select most commonly used search terms. We did not use the term ‘software development’ in our search since it was too specific and limiting, even though it is the context in which we are working. We do not aim to compare methods and therefore did not use the comparison criteria nor did we use the outcomes criteria since the scope of the outcomes of this study is hard to limit. A pilot literature search was then performed to verify whether omission or addition of one or more search terms could lead to a decrease or increase in the documents returned. Search terms that did not add any new documents were dropped and others kept. After several iterations, the following search string was defined:

\[
\text{TITLE-ABS-KEY}(\text{agile OR agility OR "continuous integration" OR "continuous deployment" OR "continuous delivery" OR scrum OR "agile practices" OR xp}) \text{ AND TITLE-ABS-KEY("safety-critical systems" OR "safety-critical software" OR safety OR "safety critical" OR regulated OR regulation OR "software intensive")}
\]

3.2.2 Inclusion and Exclusion criteria

Using the search string obtained, we performed an automated search on Elsevier Scopus (www.scopus.com), a database indexing many relevant venues and journals. The search returned 1986 papers which were then limited to the 'Computer Science' subject in Scopus leaving 546 papers. From these, all duplicates, editorials and panel paper were removed leaving 465 papers. The papers that were included were those that presented any kind of empirical study dealing with application of agile methods in a safety-critical environment, which were written in English and which were published from 2001, when the agile manifesto was launched, to 2017.

On the other hand, studies were excluded if they were pure discussion and opinion papers, duplicates (e.g. conference paper extended into journal), or if their main focus was not agile development in SCS. A summary of the selection criteria and as well as the percentage of the paper excluded and included based on our criteria are shown in Table 3.1.

After removing the 6 non-english papers, the first author went through the titles and abstracts of all remaining 458 papers to determine their relevance for the study. The titles and abstracts were taken together since from the preliminary search, we realised that some studies’ titles are not that explicit and could lead to elimination of relevant studies. In this step 391 papers were excluded, since their title and abstract were clearly not about agile software development and SCS. In order to increase reliability of our study, we had a random sample of 20 papers rated by three of the other authors and computed an inter-rater agreement (Fleiss Kappa $\kappa = 0.595$). All disagreements were discussed and resolved among the researchers, leading to an agreed result of 69 papers. Then, the first two authors went through the full texts of these remaining 69 papers collaboratively and selected 33 papers based on the inclusion and exclusion criteria in Table 3.1, to which we added one more paper identified through snowballing \[107\] for consideration (indicated with +1 in Table 3.1), resulting in a total of 34 papers.
### Table 3.2: Table: Data extraction template

<table>
<thead>
<tr>
<th>Data item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>The title of the paper.</td>
</tr>
<tr>
<td>Authors</td>
<td>The full names of the authors.</td>
</tr>
<tr>
<td>Year</td>
<td>Year of publication.</td>
</tr>
<tr>
<td>Venue</td>
<td>Name of publication venue.</td>
</tr>
<tr>
<td>Publ. type</td>
<td>Journal, conference or workshop.</td>
</tr>
<tr>
<td>Summary</td>
<td>The main content of the paper.</td>
</tr>
<tr>
<td>Benefits</td>
<td>The benefits of agile development in SCS.</td>
</tr>
<tr>
<td>Challenges</td>
<td>Problems faced with agile development in SCS.</td>
</tr>
<tr>
<td>Practices</td>
<td>The practices used in that safety + agile environment.</td>
</tr>
<tr>
<td>Principles</td>
<td>The principles they follow if any.</td>
</tr>
</tbody>
</table>

#### 3.2.3 Data extraction and Synthesis

With respect to our research questions, we then extracted metadata, practices used, principles followed, challenges faced, and (potential) benefits from the 34 selected papers based on a predefined template for data extraction presented in Table 3.2. This template allowed us to extract all the information we need for analysis into a MS Excel spreadsheet. The data obtained was analyzed both quantitatively and qualitatively. We used the themes agreed during the workshop with company experts, while basing a thematic approach, to group the results obtained into the upfront, just-in-time and infrastructure. For synthesis, the metadata was summarised and proposed benefits, challenges and potential solutions were collated, analysed, and categorized among the researchers.

#### 3.2.4 Limitations and Threats to Validity

Our literature search was limited to the Elsevier Scopus database. Even though Elsevier claims that Scopus is the most comprehensive database of peer-reviewed research abstracts[1], additional databases could have provided more relevant studies. However, this was mitigated by snowballing, where the citations of the included papers were reviewed for consideration, leading to one additional paper. In order to mitigate potential misinterpretations of papers, we relied on more than one researcher in each key step of our research. Where appropriate, we calculate inter-rater agreement. Despite these efforts, we may have overlooked relevant papers or misinterpreted them in our synthesis. We aimed for a transparent description of our method to allow for recovery if this should be the case.

---

3.3 Findings

3.3.1 RQ1: Existing research about agile development of SCS

Based on our search terms in combination with the defined inclusion and exclusion criteria, we selected 34 papers\(^2\). As shown in Table 3.3, we did not select any paper before 2006 and there has been a significant increase of papers (\(\cdots\)) in 2016 and 2017, supporting our assumption that agile for SCS becomes increasingly important. While we did find a number of articles in journals and magazines (5) as well as conference papers (22), most of them are quite short. Together with a considerate number of workshop papers (7) this suggests that the research field is still young. In Table 3.3 we also give a rough characterization of the main concern of the selected papers of each year, based on concepts highlighted in the abstracts and keywords. While the first papers in 2006-2010 were very positive about agile methods, consecutive papers discuss more and more difficulties. From 2011 on, papers aim to systematically analyse and overcome such obstacles. We observe proposals to enhance (plan-driven) methods for developing SCS with agile practices, but also suggestions to enhance agile methods with concepts from safety standards.

3.3.2 RQ2: Key benefits of applying agile methods to SCS

With respect to RQ2, we derived key benefits from the papers and grouped them into the following categories:

Efficient use of available information: Agile methods advocate responding to change over following a plan and encourage developers to use available information to start or continue their work \[109\]. Even early sprints aim to deliver working software which helps to identify and address technical risks, e.g. with respect to safety cases, early \[110\]. This promises to remove the need for heavy upfront design \[111\].

Improved stakeholder involvement: In SCS development, coordination and understanding between the relevant stakeholders is of utmost importance. Agile methods enable continuous communication between different stakeholders \[112\], thus allowing them to contribute to development \[109\]. This keeps the team better aligned with stakeholders and ensures that user needs and intended use are well understood \[110\], increases customer orientation \[113\], and strengthens the trust between stakeholders \[114\].

Improved safety culture: In an agile setting, the available information can be used to get an early start on the safety analysis \[109\]. This helps in identifying possible barriers to safety which also makes it easier to discover and correct faulty system requirements \[112\]. The practice of frequent integration of software and hardware elements supports early identification of potential issues \[115\]. The early focus on safety improves the level of safety awareness within the team and among stakeholders, thus improving the safety culture.

Improved management of changing requirements: Agile methods tend to focus on features during development. Together with the iterative nature of

\(^2\)All papers: [https://docs.google.com/spreadsheets/d/19e4a7y38f0y38mY3h1F171q725a2D12Y8vCQQC/edit?usp=sharing](https://docs.google.com/spreadsheets/d/19e4a7y38f0y38mY3h1F171q725a2D12Y8vCQQC/edit?usp=sharing)
### Table 3.3: Overview of Papers

<table>
<thead>
<tr>
<th>Year</th>
<th>Ct.</th>
<th>Journal</th>
<th>Conf./WS</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017</td>
<td>5</td>
<td>EuroSPI [132], ICSE [117], Profes [123], XP [113], XP WS [124]</td>
<td></td>
<td>Hazard analysis, Safety Assurance, Fitting Safety into Scrum, S-Scrum, Documentation, Safety Story/Epic, Continuous Safety-Builds, Continuous Delivery</td>
</tr>
<tr>
<td>2015</td>
<td>1</td>
<td>ICCNNEEE [133]</td>
<td></td>
<td>systematic iterative approach, safety modular decomposition, safety argument</td>
</tr>
<tr>
<td>2014</td>
<td>4</td>
<td>eChallenges [126], SAFECOMP [116], SPICE [118], XP [114]</td>
<td></td>
<td>agile (in-process) change impact analysis, hybrid: plan-driven and agile, documentation in agile, quality assurance in agile, compliance by design</td>
</tr>
<tr>
<td>2013</td>
<td>4</td>
<td>BI&amp;T [110] ICSE [67], Profes [100], SERENE WS [134]</td>
<td></td>
<td>human factors, good documentation, certification cost, component reuse, iterative processes, agile and regulation, conformance</td>
</tr>
<tr>
<td>2012</td>
<td>4</td>
<td>(CSCI [119])³</td>
<td>SPICE [127], FormSERA WS [121], ISSREW [125]</td>
<td>risks of agile, documentation, traceability, regulatory compliance, up front planning, managing multiple releases, formal methods, requirements analysis, change management</td>
</tr>
<tr>
<td>2011</td>
<td>2</td>
<td>LJCCBS [122], SPE [136]</td>
<td></td>
<td>can agile processes be applicable to SCS? empirical process control, right amount of ceremony</td>
</tr>
<tr>
<td>2010</td>
<td>2</td>
<td>AGILE [101], LESS [104]</td>
<td></td>
<td>minimal up front design, iterative development, Lean/Agile adoption for SCS, hybrid processes</td>
</tr>
<tr>
<td>2009</td>
<td>1</td>
<td>AGILE [115]</td>
<td></td>
<td>agile is best suitable for FDA regulated medical devices</td>
</tr>
<tr>
<td>2008</td>
<td>1</td>
<td>AGILE [120]</td>
<td></td>
<td>test automation, requirements validation, reporting, performance metrics</td>
</tr>
<tr>
<td>2006</td>
<td>1</td>
<td>XP [131]</td>
<td></td>
<td>agile helps with changing requirements of SCS, effectiveness of agile practices changes over time</td>
</tr>
</tbody>
</table>

³) Paper added through snowballing since not included in scopus results.

Agile methods, this significantly reduces requirements churn [115] and provides teams with the ability to manage new and/or changing requirements [113]. While most safety requirements may not change that often, it is important to understand the impact a change, for example in functional requirements, would have on safety. This enables teams to re-plan their sprints basing on the most recent understanding of the system under development and its requirements [112].

**Improved prioritisation:** Agile methods generally give the advantage of better prioritisation during the handling or managing of changing requirements [116]. With prioritisation, focus is put on the highest value feature first, which leads to more thorough definition and validation of the important requirements [110]. This allows to put safety related features into the center of attention when developing SCS.

**Mapping of functional and safety requirements:** Studies propose maintaining
3.3. FINDINGS

specific backlogs for safety and functional requirements such that the development takes safety into context from the start. This practice supports a mapping between the functional and safety requirements and seamless integration of safety and software engineering.

Reduced costs: Agile methods encourage simple designs which in turn produce simpler software with reduced development and maintenance cost. The high response to requirements’ changes that agile methods advocate helps to reduce the amount of rework and shortens overall time spent on development as well as the cost that would be spent on redoing the work. High development costs are a big challenge in developing SCS and the ability to reduce cost and lead time for bringing new safety-critical functions on the market will allow to increase the speed at which organizations learn on how to make such functions even safer.

Better test cases: The iterative development allows the team to only prepare and maintain documents that are needed either for development or certification. This also produces a lower number of tests, but with better focus on their intent, making it easier to understand and debug.

Improved quality: SCS development is a quality matter which is usually checked through rigour in testing. In agile methods, the continuous availability of working software and early response to changes facilitates ongoing testing, which reduces the risk to product quality. Rottier and Rodrigues report that the use of iterative development together with Test Driven Development (TDD) produced fairly consistent high quality code for their product.

Improved opportunities for reuse: Good adaption of agile methods helps building new frameworks that can be reused in later phases or projects, which will improve efficiency. Such frameworks can also enable the reuse of test cases and reduce the overall workload.

3.3.3 RQ3: Challenges with agile development of SCS

Several of our selected papers identified challenges relating to use of agile methods in SCS development, which we group and summarize into the following themes:

Difficult to manage knowledge flow between many stakeholders: There are many stakeholders in SCS development, including other engineering disciplines, sub-system suppliers, users, and external bodies such as certification authorities. Managing communication among these stakeholders is challenging. While agile methods may improve stakeholder involvement (see above), stakeholders of SCS are used to communication through documents. The lack of focus on documentation in agile methods may lead to unclear safety documentation, thus hindering effective communication between the different stakeholders.

Safety standards facilitate a waterfall mindset: Safety standards are prescriptive and often described with a waterfall-based process in mind. With this mindset, regulated domains often value effectiveness of waterfall processes over the flexibility of agile methods. For example, many safety standards demand that one person must not create and review the same artifact, while agile methods favor cross-functional teams. This
mismatch of the mindset causes risks to potential compliance \[110\].

The waterfall mindset also shows a strong focus on documentation. In SCS, documentation is required for showing regulation compliance \[127\] and is the preferred way of communication with certification bodies \[122\]. Documentation is also the primary evidence for traceability \[67\]. Agile methods are often reported to suffer from this focus on documentation \[67\,114\,122\,124\,126\,128\] as extensive documentation will diminish advantages of agility while lack of documentation will lead to insufficient traceability.

**Lack of trust in agile methods:** Since agile methods do not give clear guidelines for project monitoring and control, it is hard to determine the sufficient level of evidence to present \[129\] for certification. Thus, it appears difficult to estimate effort and through the “unstructured nature” of agile methods, the amount of effort for delivering a feature seems to be underestimated \[120\]. The maturity of agile methods is hard to compare to CMM and ISO standards \[122\] and many managers are not convinced about business benefits of agile methods \[67\]. There is also doubt about whether agile methods can provide sufficiently rigorous testing \[122\].

**Upfront planning:** Safety standards suggest upfront design, hazard identification, and analysis. With agile methods, it is hard to determine how much time is enough for upfront planning \[123\,127\] and consequently difficult to identify all of the ways in which software can contribute to system level hazards up front \[129\]. Thus, there is the perception that planning in agile methods is insufficient for SCS development \[67\,124\]. The aim for short upfront design and for analysing requirements just in time during iterations puts time pressure for determining the safety requirements and makes it difficult to evaluate the quality of the safety arguments \[101\,123\] which could impede certification \[119\]. Even more, attempts to include safety in agile can shift focus from customer value towards verification and validation efforts \[130\].

**Flexibility vs. Safety:** Agile methods do not provide practical guidelines for change impact analysis \[116\], yet they provide flexibility in managing requirements. Every update therefore calls for relentless testing and strict configuration management \[113\] and un-coordinated software changes can lead to increased complexity as the project progresses \[131\]. In addition, it is difficult to manage requirements that change iteratively \[114\].

### 3.3.4 RQ4: Solution candidates (e.g. principles and practices) for challenges with respect to agile development of SCS

We present solutions collected from the selected papers in Table 3.4. Some of these solutions relate to enhancing agile, others to complementing safety, which emphasizes the aim of these works to bring both worlds closer together. We organized the proposed solutions in different categories, starting with principles that were suggested to provide guidance for agile development of SCS. These are rather abstract guidelines or goals to strive for when implementing an agile way of working with SCS. In addition, we identify proposed solutions with respect to process and release planning and roles.

We then continue with more concrete practices that were proposed in literature, which we grouped in relation to testing and continuous development,
Table 3.4: Solution Candidates in Literature

<table>
<thead>
<tr>
<th>Solution cand.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Principles</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Customer involvement at all levels</strong></td>
<td><strong>On-site customer</strong> should be part of hazard analysis, safety analysis, SSRS requirements phase, sprint reviews at all levels of product development. <strong>Product owner</strong> can act as the on-site customer. <strong>Frequent contact with the customer</strong> hand regular communication allows team to react on the specific needs of the customers.</td>
</tr>
<tr>
<td><strong>Risk management</strong></td>
<td>Identify high level risks in initial phase and consider risks during development. Ensure that software is truly safe by employing a risk-based approach to planning, testing, verification and validation.</td>
</tr>
<tr>
<td><strong>Process and Release planning</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Iterative / incremental development</strong></td>
<td>Develop components iteratively in fixed and short iterations delivering functional software. Do iterative safety analysis, have an incremental safety validation plan and incremental safety case.</td>
</tr>
<tr>
<td><strong>Backlog management</strong></td>
<td><strong>Team should maintain</strong> a groomed, refined, and prioritised backlog with two parts: one for functional requirements and one for safety requirements.</td>
</tr>
<tr>
<td><strong>User stories</strong></td>
<td>Used for product definition, high level requirements, upfront planning, and as basis for safety analysis. Refine with use case diagrams or textual use cases.</td>
</tr>
<tr>
<td><strong>Essential upfront plan</strong></td>
<td>Do architectural design and hazard/safety analysis upfront. Use hazard lists or checklists from literature; perform analysis of previous failure reports for process hazard analysis on the user stories. Use FMEA for decision support.</td>
</tr>
<tr>
<td><strong>Formal change control and prioritisation</strong></td>
<td>Implement formal change control process with upfront lightweight CIA to assign quality scores to all requirements and to prioritize requirements. Maintain CIA report (issues raised and resolved). <strong>Formalise design reviews</strong> for validation and verification.</td>
</tr>
<tr>
<td><strong>Roles</strong></td>
<td>Teams are self-organized and empowered to manage daily tasks of producing software on their own. Rely on collective code ownership to allow the whole team to help and keep track of who did what.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 3.4 – continued from previous page

<table>
<thead>
<tr>
<th>Solution cand.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expert knowledge</td>
<td>Include expert knowledge in team [119,134]: QA [132] or safety experts [111,123,130] or nominated safety team member [129].</td>
</tr>
</tbody>
</table>

**Testing and Continuous Development**

| Reviews | A team of peers with assigned roles performs code reviews \[67,136\] or more formal technical reviews \[133\]. Apply Test driven development (TDD) to establish high test coverage \[68,104,111,113,120,121,125,131\] and increased regression testing \[135\]. Tests should be automated \[120,135\], risk-based \[126\], continuous and extensive \[119,136\], and should include safety tests \[137\]. |
| Acceptance and unit testing | Do unit tests \[115,131\] and acceptance tests \[116,122,131,134\] once a number of iterations have been completed \[127\]. |
| Continuous Build | Apply continuous integration \[67,68,110,113,122,125,126,131\] supported by automation \[115\] and include continuous safety builds \[117\]. Strive for continuous compliance \[67\] in order to facilitate continuous delivery \[67,113\]. |

| Coding standard | Keep high coding standards that all developers should adhere to \[113,125,134\], supported by pair programming \[104,114,121,122,125,131\] and refactoring \[68,114,122,125\], e.g. with help of refactoring stories \[67\]. |

**Regular meetings**

| Daily meetings | Daily meetings \[67,68\] to provide visibility of safety requirements satisfaction status \[129\] and promote collective ownership \[115\]. Daily stand-ups \[67\] provide feedback, communication, and coordination to manage technical and organizational dependencies \[131\]. Establish weekly goals \[67\] and review them at the end of each week during the daily meeting to focus on team objectives \[115\]. |

| Sprint review | Demonstrate to product owner what has been done by the team \[67,68,115,121,124,135\] and to create awareness of activities among team members \[121\]. Include all relevant stakeholders \[129\]. Include hazard analysis (ibid) and acceptance criteria in review \[110,118,124\] and implement it as independent process for good quality assurance \[126\]. |

| Planning meetings and retrospectives | Create software development plan \[126\] based on features \[115\] at the planning meeting \[68,116\]. Document chosen modelling approach, implementation language, development environment and assessment tools \[129\] and determine safety requirements \[123\]. Combine sprint planning with retrospective \[109,115,135\] to empirically improve estimations \[67\]. |

**Safety Engineering Practices**

Continued on next page
Table 3.4 – continued from previous page

<table>
<thead>
<tr>
<th>Solution cand.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traceability support</td>
<td>Use Traceability to support certification as well as claims that requirements have been met. Use appropriate tools for automation. Use trace information to determine which tests need to be rerun.</td>
</tr>
<tr>
<td>Standard Operating Procedures (SOPs)</td>
<td>Rely on SOPs to improve quality management with clear guidelines to developers and use them as management records for achieving some compliance objectives in agile. The records should include safety standards the team should be familiar with and short written policies for documentation, review and testing, developed by the team.</td>
</tr>
</tbody>
</table>

3.3.5 Synthesis of Findings

In Table 3.5 we give an overview of key benefits, challenges, principles, and practices discussed in literature and in Fig. 3.1 we show how the selected papers are distributed over these categories. The majority of work relates to just-in-time activities, such as analysis and management of functional or system safety. Fewer selected papers relate to upfront analysis, which in many publications is taken for granted. The smallest amount of papers relates to long-term aspects and infrastructure that lasts beyond an individual project.

It is noteworthy that solution candidates dominate for upfront and just-in-time aspects. In contrast, we found relatively fewer papers relating to solutions with respect to long-term and infrastructure aspects, especially in relation to the amount of challenges found in this category. We found only few papers that discuss a trade-off between upfront analysis and just-in-time analysis or attempt to push more activities from the upfront phases into continuous development.

<table>
<thead>
<tr>
<th>Benefits (RQ2)</th>
<th>Challenges (RQ3)</th>
<th>Solutions (RQ4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>12</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>31</td>
</tr>
</tbody>
</table>

Figure 3.1: Bubble chart with numbers of papers per category.

This result resonates well with the discussions during our initial workshop. Our industry participants highlighted the following challenges that we did not yet found sufficiently supported in the selected papers.

**Upfront:** Our industry participants saw benefits in upfront analysis for agile development of SCS, e.g. with respect to reducing dependencies and confining safety-critical or secret software (e.g. due to IP constraints).
Table 3.5: Summary of literature

<table>
<thead>
<tr>
<th>Benefits</th>
<th>Total Pri. Papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improved stakeholder involvement</td>
<td>5</td>
</tr>
<tr>
<td>Reduced costs</td>
<td>5</td>
</tr>
<tr>
<td>Improved quality</td>
<td>4</td>
</tr>
<tr>
<td>Efficient Use of available information</td>
<td>3</td>
</tr>
<tr>
<td>Improved safety culture</td>
<td>3</td>
</tr>
<tr>
<td>Improved opportunities for reuse</td>
<td>3</td>
</tr>
<tr>
<td>Improved management of changing requirements</td>
<td>3</td>
</tr>
<tr>
<td>Improved prioritisation</td>
<td>2</td>
</tr>
<tr>
<td>Mapping of functional and safety requirements</td>
<td>2</td>
</tr>
<tr>
<td>Better test cases</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Challenges</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong focus on documentation</td>
<td>9</td>
</tr>
<tr>
<td>Upfront planning</td>
<td>9</td>
</tr>
<tr>
<td>Safety standards to facilitate a waterfall mindset</td>
<td>6</td>
</tr>
<tr>
<td>Lack of trust in agile methods</td>
<td>5</td>
</tr>
<tr>
<td>Flexibility vs Safety</td>
<td>4</td>
</tr>
<tr>
<td>Difficult to manage knowledge flow between stakeholders</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Practices and principles</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceptance and unit testing</td>
<td>18</td>
</tr>
<tr>
<td>Coding standard</td>
<td>14</td>
</tr>
<tr>
<td>Continuous build</td>
<td>11</td>
</tr>
<tr>
<td>Backlog management</td>
<td>11</td>
</tr>
<tr>
<td>Iterative development</td>
<td>10</td>
</tr>
<tr>
<td>Sprint review</td>
<td>10</td>
</tr>
<tr>
<td>Planning meetings and retrospectives</td>
<td>9</td>
</tr>
<tr>
<td>Formal change control and prioritisation</td>
<td>8</td>
</tr>
<tr>
<td>Expert knowledge</td>
<td>7</td>
</tr>
<tr>
<td>Customer involvement at all levels</td>
<td>7</td>
</tr>
<tr>
<td>User stories</td>
<td>5</td>
</tr>
<tr>
<td>Essential upfront plan</td>
<td>6</td>
</tr>
<tr>
<td>Essential daily meetings</td>
<td>5</td>
</tr>
<tr>
<td>Self-organising teams</td>
<td>4</td>
</tr>
<tr>
<td>Traceability support</td>
<td>3</td>
</tr>
<tr>
<td>Risk management</td>
<td>3</td>
</tr>
<tr>
<td>Reviews</td>
<td>2</td>
</tr>
<tr>
<td>Standard operating procedures</td>
<td>2</td>
</tr>
</tbody>
</table>

Challenges relate to a lack of experience with continuous development of existing products in contrast to developing new products.

Just-in-time: Industry participants saw benefits with respect to managing continuously changing dependencies and requirements. The ability to do continuous assessment and certification was seen as a real enabler. Also, agile documentation, i.e. focusing on the essential, product-related information was seen as a potential benefit. Challenges relate to constructing the big picture from local information, quality assurance and assessors with waterfall mindset, and the question whether product or process based evidence for safety would be the better choice. With respect to solutions, they showed big hopes in applying specification by example and hardening sprints.

Long-term/infrastructure: In this category, our participants saw the biggest need for new concepts. Potential benefits could include investment in infrastructure with long-term benefits, organization-level assessment, tools to check dependencies, and semi-automation for traceability. The biggest challenges were seen in a pay-per-product mentality, lack of traceability, too many layers of requirements (making it impossible to keep safety requirements up-to-date in continuous development), certification of tools (which will slow down their
evolution), old assessment frameworks, and lack of trust in agile maturity.

3.4 Discussion and Conclusion

In this paper, we provide a systematic mapping of agile development of SCS. We contribute an overview of research papers discussing experience from industry from 2001 to 2017 in the hope that this will enable future research in this area. Our synthesis is based on iterative analysis of selected papers and discussions in a workshop with experts from six major Swedish product companies. We found that potential benefits, challenges, and proposed solutions relate to upfront, just-in-time, or long-term and infrastructural aspects. While companies continue to struggle in all three areas, we see a need for future research specifically in two areas: a) Investigation of trade-offs between effort done upfront and just-in-time together with guidelines on how to shift more effort into just-in-time analysis; b) Investigation on how to establish beneficial infrastructure for long-term support.

Acknowledgements: We thank all participants in our workshop for their great support, deep discussions, and clarifications throughout this work whenever needed. This work was supported by Software Center (www.software-center.se) and the SIDA BRIGHT project.
Chapter 4

Paper C

Agile Islands in a Waterfall Environment: Challenges and Strategies in Automotive

R. Kasauli, E. Knauss, J. Nakatumba-Nabende, B. Kanagwa

*In Evaluation and Assessment in Software Engineering (EASE’20), 2020.*
Abstract

Driven by the need for faster time-to-market and reduced development lead-time, large-scale systems engineering companies are adopting agile methods in their organizations. This agile transformation is challenging and it is common that adoption starts bottom-up with agile software teams within the context of traditional company structures. This creates the challenge of agile teams working within a document-centric and plan-driven (or waterfall) environment. While it may be desirable to take the best of both worlds, it is not clear how that can be achieved especially with respect to managing requirements in large-scale systems. This paper presents an exploratory case study focusing on two departments of a large-scale systems engineering company (automotive) that is in the process of company-wide agile adoption. We present challenges that agile teams face while working within a larger plan-driven context and propose potential strategies to mitigate the challenges. Challenges relate to, e.g., development teams not being aware of the high-level requirements, difficulties to manage change of these requirements as well as their relationship to backlog items such as user stories. While we found strategies for solving most of the challenges, they remain abstract and empirical research on their effectiveness is currently lacking.
4.1 Introduction

In order to meet market demands and increasing competition, large-scale systems engineering companies are adopting agile methods [44, 54, 137]. To leverage the full benefits of agility with respect to time-to-market and flexibility, such agile transformation must in the long run affect the overall systems engineering process and organization. While software teams may have quickly adopted agile methods and work with them, there is usually slow company-wide adoption, mostly attributed to the skepticism they have received [13]. Since agile teams have to work with the traditional structures within the companies, this ends up creating ‘agile islands in a waterfall’ [138] – a coexistence of both agile and traditional development approaches.

However, without a company-wide adoption structure, different development teams adopt different agile practices [139]. Further, in the systems engineering context, software development is only one of several domains and must be synchronized with development of hardware and mechanics. With software teams contributing to the same product, it becomes challenging to deal with different speeds and quality measures from the different ways of working. While there is a growing body of literature on agile transformations [8, 140, 141], research addressing the coexistence of agile and traditional methods is only recently growing [36, 37, 39, 137, 142]. We are not aware of any empirical work that explores such coexistence from the development teams’ perspective, especially in relation to the specific way teams adopt agile practice.

This study addresses that gap by first understanding and documenting the challenges and lessons learned based on a case study situated in the automotive domain. This domain is particularly interesting, since requirements traditionally play an important role in automotive systems, yet recent trends around electrification, advanced driver assistance systems (ADAS), and autonomous drive (AD) as well as a market that moves towards continuous deployment of software functions render it necessary to change established engineering practices.

Through an exploratory case study with two departments of an automotive company, based on a total of 18 interviews and one focus group, we explore the following research questions:

- **RQ1:** What are the perceived challenges when combining plan-driven and agile paradigms in large-scale systems engineering?

- **RQ2:** What mitigation strategies exist for the challenges in RQ1?

In answering our research questions, we provide important empirical data about the interplay of agile software development teams operating in a larger, plan-driven system development context from a requirements engineering perspective. We present specific challenges with their proposed mitigation strategies. We believe that this study will benefit both researchers and practitioners working in companies with similar environments.
Related Work

Traditional methods like waterfall and V-model have formed the foundation for systems development for decades. These methods follow a sequential execution of processes with predefined phases, extensive requirements design and documentation [39,137]. In traditional methods, RE includes a set of well defined preliminary phases dealing with analysis, planning, and documentation [143]. Meaning that requirements are supposed to be complete in the preliminary stage. However, faced with evolving market demands and fast changing requirements, systems development companies are adopting agile methods [36,37,138] as the traditional methods are not flexible and are generally slow due to their rigorous nature.

Agile methods encourage flexible and light-weight software development with short iterations [30]. Being flexible, agile methods can help where traditional methods fail [40]. In agile development, RE is an iterative and continuous process that is carried out in every step of development. Also, requirements are only partially known and evolve rapidly during development. Previous studies on agile and traditional development have compared the characteristics of agile development with those of traditional methods [33–35,144] while describing the way different organizations are adopting to these methods. There are few studies concerning the coexistence of agile methodologies with traditional approaches [36,37,39,40,137,142,145].

Theocharis et al. [37] studied the general process use over time to investigate how traditional and agile methods were used. They aimed to find out if there was coexistence or whether agile methods accelerated the traditional processes’ extinction. This was done by applying instruments of systematic literature review process. They found indication of mixed application of traditional and agile methods and thus concluded that hybrid approaches that include traditional and agile approaches shaped today’s “standard process ecosystem”. The results of their exploratory study show that there is less material published on the combination of software development approaches than expected. This is supported by Kuhrmann et al. [36,145] who present results from 69 study participants in a survey on hybrid software development approaches. They found that companies combine different development approaches regardless of the industry sector and size. These findings go beyond the adoption problem to having a state of working with both methods in coexistence. This confirms the coexistence of both methodologies.

Based on a Grounded Theory study involving 21 agile practitioners from two large enterprise organizations in the Netherlands, Waardenburg and van Vliet [137] presented challenges of using agile methods in traditional enterprise environments. They organized the challenges under two factors: increased landscape complexity and lack of business involvement, for which they identify successful mitigation strategies. These mitigation strategies concern the communication between the agile and traditional part of the organization, and the timing of that communication. Kuusinen et al. [40] investigate practitioners’ mitigation strategies related to the challenge of doing Agile in a non-Agile environment. Through an on-line survey and a workshop, they provide strategies from both an organizational and change perspective. The strategies include: (a) ensuring managers understand and buy-in to Agile and (b) creating an organi-
izational culture that fosters agility as the biggest themes of the study. Whereas both these studies give the organizational view, we provide the developers’ perspective.

Theobald and Diebold 142 based on a literature search and workshops to identify existing interfaces of agile to non-agile environments for which they collect and group problems. They propose a classification matrix based on which we describe our research here to relate to the “project-team interface” and to provide deeper empirical knowledge about the related challenges and problem areas. Starting from a literature search, Kusters et al. 39 derive risks and problems at the interface of agile and traditional development approaches in hybrid organizations which have an impact on coordination and cooperation. They discover 28 issues which reduce to 22 after validation through a case study at a large financial institute in the Netherlands. They had six classifications of challenges including challenges relating to development and testing. We explore challenges in this classification, particularly software development.

In summary, the body of knowledge on the coexistence of agile and traditional methods is growing. However, to the best of our knowledge, there is a lack of empirical studies that explore developers’ experiences in this context. We believe this study will help inform a better approach to the coexistence dilemma.

4.3 Research Method

In this study, we explore development challenges caused by the interplay of agile and plan-driven methods and a lack of a company-wide strategy for agility. We rely on the case study method 76, which is considered appropriate when studying a complex social contemporary phenomenon for which deep understanding of the context is critical 146. Specifically, we investigate the case of an automotive company and using two of its departments as units of analysis. We collected data through 18 semi-structured interviews and used thematic coding for analysis. This section details our data collection and data analysis steps and also discusses the validity threats.

4.3.1 Data Collection

The study was conducted at two different departments (A and B) of a large automotive manufacturer (OEM) distributed across several countries. The departments are among those that pioneered the company’s increased move to in-house software development with the aim to further increase flexibility and to decrease the lead-time for late changes.

We started with a study at Department A, whose results motivated the replication of a similar investigation at Department B. The research process is depicted in Figure 4.1.

Semi-structured interviews were conducted at both departments with 11 interviews at Department A and followed up with a focus group meeting. The focus group meeting consisted of some of the interview participants and other new members. At Department B, seven interviews were conducted and followed up with emails and phone calls for failure to agree on a physical meeting time.
Throughout the different phases of this research, the authors collaborated in selecting participants and defining the interview guide based on consultations with our main contacts in both departments. Participants were selected from two role categories: a) the agile software teams and b) the plan-driven system level roles. Both categories think very differently about development: the agile software teams did not necessarily like to talk about requirements, while the system level roles did not feel that a discussion of agile artifacts would provide value. This was mainly a problem at Department A, where we chose to avoid using the word “requirements” in our interview guide for agile software teams, where necessary. In Department B, this aspect was deemed less critical, but since the way of working differed significantly, we had to adjust the interview guide again for their context — allowing us to follow up on findings from Department A. We discuss each department in more detail in section 4.4.

We used the research questions given in Section 4.1 as guiding questions during our investigation and refined them while analyzing the data. All interview guides contained questions which sought information on how requirements flow from one role to another within the development teams. Each interview was started by explaining the purpose of the interview. Interview time ranged from 45 min to 1 hour. All interviews were voice-recorded and notes were taken during the interviews. The roles and responsibilities of interviewees for each of the departments are presented in Table 4.1. Involving the different roles gave us the opportunity to explore the different interpretations of the phenomenon.

### 4.3.2 Data Analysis

The interviews were transcribed and thematic coding method [78] was used to identify, analyze and report patterns within the data. Transcriptions were used to identify, name and categorize phrases and words in order to develop the initial codes. Interviewers also provided initial codes, enabling the generation of inductive codes which were later grouped into themes. The themes were iteratively and collaboratively refined by all authors and named according to the aspects addressed in the interview guide. We thus agreed on the following general themes: requirements engineering process (which included the teams involved and roles), challenges faced, and solutions proposed.

---

1 Interview guides: [http://www.cse.chalmers.se/~knauss/2020-AgileIslands](http://www.cse.chalmers.se/~knauss/2020-AgileIslands)
### Table 4.1: Roles and responsibilities of participants in the study.

<table>
<thead>
<tr>
<th>Role</th>
<th>Responsibilities</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function Owner (FO)</td>
<td>Owner of one or several car functions and produces the high-level requirement(s) for the function.</td>
<td>2</td>
</tr>
<tr>
<td>Function Realisator (FRR)</td>
<td>Breaks down the FO requirement to a slightly more detailed description of how the function should be realized and distributes to different sub-systems.</td>
<td>1</td>
</tr>
<tr>
<td>System Designer (SDE)</td>
<td>Describes the high-level requirement and design of the sub-system.</td>
<td>1</td>
</tr>
<tr>
<td>System Responsible (SR)</td>
<td>Produce a detailed design of the software and hardware and also write detailed requirements (SRS).</td>
<td>1</td>
</tr>
<tr>
<td>Software Quality Engineer (SQE)</td>
<td>Comes in when something goes wrong to ensure that correct design is followed</td>
<td>1</td>
</tr>
<tr>
<td>Testers (SFT)</td>
<td>Depending on what level the tester is, they are responsible for writing a test &amp; verification plan that describe the verification steps to cover the requirement, and then perform verification/validation and report the results.</td>
<td>2</td>
</tr>
<tr>
<td>Scrum Master (SM)</td>
<td>“Responsible for ensuring the team lives agile values and principles and follows the processes and practices that the team agreed they would use” [147].</td>
<td>1</td>
</tr>
<tr>
<td>Product Owner (PO)</td>
<td>Manages and prioritises backlog.</td>
<td>1</td>
</tr>
<tr>
<td>Developer (Dev)</td>
<td>Create Software solution.</td>
<td>2</td>
</tr>
</tbody>
</table>

### 4.3.3 Threats to Validity

Three researchers worked iteratively with codes and their grouping into themes, during which potential misinterpretations were raised and discussed. In order to increase conclusion validity, the initial results were discussed in a focus group at Department A that included four roles already involved in the study and two more people responsible for requirements management. Results were presented in form of PowerPoint slides and discussed to check whether we misinterpreted any data and to provide an opportunity to raise any challenges we may have missed. We then discussed strategies to overcome the challenges with the participants. For Department B, we shared a PDF version of the findings through email and when clarifications were needed, calls were made. To increase internal validity, we used data triangulation between the case departments. We take two departments within one company in an in-depth qualitative exploratory case study and through sufficient numbers of interviews and member checking, we made sure that we captured all important concepts in scope of our inquiry. Both cases however yield slightly different results as we will discuss in this paper. As a first exploration, we believe that this will enable future research in the area, and we deliberately choose a research method
which sacrifices generalization of our findings to other domains in favour of in-depth-knowledge in the specific case. We believe by discussing the results with company contacts in both departments and by comparing them to existing problems in literature, we mitigate threats to reliability of our findings.

4.4 The Case Study

Company X follows a hierarchical structure and employees are departmentalized, following a chain of command. Software is partly developed in-house and partly purchased from suppliers. Recently, however, software development is increasingly moved in-house as the company endeavors to shift from the traditional methods of working to being more agile development oriented in all working departments.

In earlier work, system engineering organizations were found to differ in ‘scope’ as they introduce agile methods [138]. While few companies have achieved some level of agility for the full organization, at the time of this research, many had only introduced agile methods on the level of agile software development teams, while keeping a plan-driven approach for the overall systems engineering process. Our case company is no exception, yet both departments covered differ in how they manage the interplay of plan-driven systems engineering and agile software development. While both departments have significant experience with agile software development to support their business goal of increased flexibility, they differ in the type of software to be developed.

Department A is responsible for development of (often safety-critical) functions of the overall vehicle’s system. Their development focuses on algorithms that connect large amounts of vehicle data with cloud intelligence, and agility helps to increase learning about how these algorithms behave in a realistic context early on (a comparable setup has been discussed in [148]). While suppliers for hardware, for example chip-sets exist, much of the software is done in-house to increase flexibility, decrease lead-time of new features, and protect innovations and intellectual property.

Department B, in contrast, is responsible for the development of a central platform that is the foundation for the work of several other departments. Here, agility is employed to prioritize change requests from many stakeholders and in this way maximize the flexibility of overall system development. Thus, Department B has potentially huge impact on flexibility and lead-time for the whole system.

In order to give context to our findings, we extracted an overview of the current RE process at both departments, which we describe in terms of the roles and responsibilities (Tab. 4.1 and Fig. 4.2).

4.4.1 Roles in the Departments

While software teams aim to increase their agility, this effort must be seen in the organizational context in which the teams are embedded. The requirements process in the two departments is executed through different roles in a strict hierarchy, with requirements going back and forth between different levels, for clarification and update where necessary. There is a lot of communication,
both formal and informal in both cases. The roles can be grouped in two categories; those that make up the hierarchy and roles that constitute the software development team.

The hierarchy levels start with the Function Owner (FO) who receives the customer function and creates a high-level document which the Function Realizer (FRR) uses to distribute requirements to the different subsystems. Depending on how scalable the requirements are, the FRR could work with the same requirements or sometimes break them down before assigning them to the subsystems. At the subsystem level, the System Responsible (SR) breaks the requirements into software (SWRS) and hardware requirements. The software requirements are sent to the in-house team while the hardware requirements are sent to the respective suppliers. At the same level, the System Design Engineer (SDE) designs the technical solution, i.e., determine how to implement the requirement and how to allocate it on different components which the in-house development team implements. The SR and SDE work together and have their own iterations as they write the requirements and respective technical solutions. This concludes the hierarchical levels.

The roles in the lower part in Table 4.1 relate to software development team as depicted at the bottom of Figure 4.2. Here the departments take different approaches to agility, (although both generally, use combinations of XP and Scrum in their development). Department A, has a Product Owner (PO) who is responsible for managing and prioritizing the backlog and then puts the requirements in Jira for implementation by the developers. In case of any change requests, the development team communicates to the PO who discusses with the FO to get the requirements changed. So the plan-driven requirements do not seem to directly interact with the agile process of development. In Department B, the SDE is the one that creates an issue in Jira which the developers use to implement the requirement. The software issue contains the ‘user story’ and the software requirement from SWRS. In case of any disagreements, the developers have to go through the SDE who then discusses with the FO, if necessary.

4.4.2 Requirements Model in the Departments

The FO produces a requirement document to specify the function and shares it with the FRR. There is handshaking of the requirement between FO and FRR through back and forth communication to verify the requirement before it is broken down and sent to the next level. Due to the iterative process, the requirement document keeps changing and the company contacts preferred to term it the requirements model. By this, they emphasise a living, evolving database of requirements and tracelinks representing the best knowledge the organization has about the current state of requirements at any given point in time.

Figure 4.2 shows what our contacts call the Narrow-V model, the hierarchical model followed by both departments. It keeps the abstraction levels from the classical V-model [149] and has three general layers of requirements, but emphasizes iterative work on system level. On the left hand side, the requirements are formed and broken down at the different levels while the right hand side shows testers that write corresponding test cases for the requirements.
The Narrow-V model is supported by different tools in the two departments. Department A uses Tool A1 for all the plan-driven levels while Department B uses two different tools; one for FO level (Tool B1) and another for FRR and SDE levels (Tool B2). For the agile part, the development part, both departments use the Jira tool to support their implementation. Anyone with access to Jira can write a requirement or user story to the developers in either case but the team only works on those approved by their immediate superiors, the PO for Department A and SDE for Department B.

The figure shows how plan-driven parts depicted in the upper part of the v-model overlap with agile parts in the lower part.

### 4.5 Challenges and Strategies

This section describes the challenges and potential solutions identified through interviews and discussions in a focus group. These challenges are presented in three parts: In 4.5.1 the challenges common to both departments are discussed while 4.5.2 and 4.5.3 discuss those challenges unique to department A and B respectively. For each challenge, we discuss the corresponding potential strategies.

#### 4.5.1 Challenges in Departments A and B

Three challenges were mentioned by more than 50% of the interviewees in both departments and these are described here.

**C.1: Development team not aware of the requirement model.**

An issue raised by developers in both departments is the lack of awareness about the FO’s requirements. In Department A, only the PO is both in direct contact
with the development team and has knowledge about the FO’s requirements. Even though the developers have access to Tool A1 where the requirements are stored, they rarely access them. This is partly due to them using a different tool in their daily work, which relates to a challenge of developers in Department B, who get their requirements from the SDE and have no access to the FO’s requirements in Tool B1. One developer mentioned not getting to know what the legal requirement is, which only the FO could know from the requirements they have written. According to developer at Department B:

“...to know that if it's a bug for example, is this something breaking the law or could I wait to implement this bug fix. Could we send this software with this bug fix or is it against the law I don't know.” — Dev-B

It is however also a challenge that the requirements, as they are currently specified, do not appear to fit the agile way of working and do not easily translate into concrete development tasks on the backlog. The differences in how FO and the development team think about requirements appear hard to bridge at the time of this case study. With the pending proposals to include (system-level) testers in the development team, it is anticipated that this challenge will grow.

**Proposed Solution:** According to the interviewees, system-level testers generally have a stronger awareness of high-level requirements than developers. Thus, one solution proposal is to have more system-level testers as part of the agile development team. However, this fusion can only succeed if the agile team members are open to discuss requirements and to increase their awareness. Also, it may be hard to find enough experienced system-level testers for each team. Thus, interviewees felt a need of forcing the agile teams to create formal test report as part of their work, where test results are related to high-level requirements, forcing them to read the requirements model. A good compromise, would be not to do that every sprint but only for important releases of the software component. Obviously, this demands for teams to get access to the requirement model, which can be an organizational challenge.

**C.2: FO over exposed to change requests.**

In a traditional hierarchical and plan-driven approach, FOs discuss requirements mainly with the FRR and few change requests are made, mostly top-down and very rarely originating bottom-up from the development team. With agile development, however, comes a higher commitment to respond to change and development teams need to take more responsibility in clarifying and refining requirements. Proposed solutions to C.1 also include increased awareness and access to requirements for the development teams. This however creates a challenge to the FOs with change requests and opinions coming more frequently and increasingly from the development teams.

In Department A, even though hierarchies do exist, everyone seems to have the right to make a change proposal to the FO. As one FO comments:

“I would say it is a good thing that many people read the requirements. But then it also means there is going to be more opinions, comments and also more work.” — FO-A

Both FOs in Department A however agree that there is a danger to become the bottleneck in this process.
In Department B, the hierarchy is stricter. For instance, developers do not get to propose changes directly to the FO, also since they do not even have access to his requirement (see C.1). Still, the FO at Department B receives many questions about the function and there is a tendency to involve the FO more in FRR and SDE level work to help making decisions about changes. The exposure to change requests may not allow time to perform other activities and FOs find themselves at risk to become bottlenecks of the process.

Proposed Solution: All Interviewees agreed that in an agile way of working, requirements should be updated bottom-up based on learning from the Sprint. A high number of change requests must be embraced by an organization that aims to fit agility into their system development. This could also help to improve requirements with arbitrary performance goals. Yet, there is no clear, non-trivial solution (like adding more resources) to remove this bottleneck. A few of the interviewees hinted on having the team more involved in updating or changing the high-level requirement and we believe that peer-reviews of such updates could help to remove bottlenecks on system level.

C.3: ‘Suffering’ Requirements traceability.

The traceability between requirements artifacts used in the two departments is suffering, with differing levels of impact for each department. In Department A, the PO writes users stories from FO requirements and the team decides how to implement them. The team members can also decide to breakdown the stories to more granular levels and put to the backlog for discussion and later prioritisation. Clearly, not all user stories need to be traced, however, it is not clear which user stories should be traced to requirements. This can lead to additional effort, when traceability must be ensured, since important information is missing and must be reconstructed. When asked to comment on traceability, one developer responded that:

“I don’t think traceability is not required or something like that. It’s just that my focus hasn’t been on documenting the function. I just focus on doing implementation and developing the function.” — Dev-A

The respondents also mentioned that it is unclear whose responsibility it is to document that traceability.

In Department B, the SDE creates a ‘software issue’ and all updates and changes affecting that requirement are tracked in that issue. A Software Requirements Traceability Matrix (SWRTM) is used to capture the implementation status of requirements and to link them to unit tests, revisions of requirements, and suggested but unaccepted changes. This document is also used by function testers.

Despite these efforts, traceability suffers from the quick pace in agile development, especially since refinements of requirements on development team level are managed in a separate tool. Thus, in both departments a lot of effort goes into maintaining high quality traces, slowing down the speed of managing changes and challenging the intended goals of transitioning to agile.

Proposed Solution: All participants agreed to make explicit to a user story whether it is relevant for tracing, but requisite that requirements model is understood. One way is to have a clear definition of ‘done’ for a given user
story, that is, a user story as ‘done’ only when it has proper information about tracing, e.g. ‘does not require tracing’ or ‘must be traced against X’.

4.5.2 Challenges Unique to Department A

C.4: All can write user stories anytime.

In agile development, user stories can typically be proposed by anyone in the development organization, although it is the PO’s responsibility to prioritise them in the backlog. Interviewees noted that although a standard process is followed once a user story arrives into the backlog, there are no proper channels for screening user stories before they get into the backlog, as anybody is allowed to write user stories. This results in several unimportant or wasteful user stories creeping into the backlog which takes up a lot of time in discussing them.

In Department B, respondents agreed that all can write user stories if they have access to the system. However, since the structure is rather strict here, developers focus to implement based on tasks received from the SDE and are generally not concerned with new user stories. Once a new user story is written, the System Team (made up of FO, SDE and SR) discusses to reconsider the requirement. If needed, the FO effects the change and developers do not get involved in managing new user stories.

Proposed Solution: There were proposals to create proper channels for user story writing, where some authentication is required before writing, something that corresponds to what is done in Department B. This might however conflict with attempts to empower the teams more to manage requirements.

C.5: Inconsistency between requirements model and user stories implementation.

The interviewees from SFT stated that they do receive all the requirements from SDE, but they do not get information on whether the requirements were not implemented or changed. So, they work on the assumption that all the prior test cases still are valid. However, during testing, they discover that either many of the requirements have not been implemented or the requirements do not match the implementation. This mismatch was attributed to failure to update requirements. The user stories change a lot during implementation and yet the traceability is not clear (see C.3). Furthermore, for the new user stories which arise during implementation, the corresponding requirements have not been written. Since traceability is lacking, requirement updating does not happen, resulting in unnecessary additional work to establish a defined state before release.

In contrast, for Department B, the traceability problem surfaces on a higher level and similar inconsistency is not so evident.

Proposed Solution: Making testers part of the development team was one strategy to overcome the challenge of inconsistency. This however entails also a cultural change and does not alleviate the problem at higher level. We believe that a way should be found to involve teams more in updating and maintaining high-level requirements (see also C.2), e.g. by making requirements an explicit part of the sprint goal.
### Table 4.2: Summary of Challenges and Proposed Strategies

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Interviewees</th>
<th>Literature</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.1 Development team unaware of requirement model</td>
<td>Have testers as part of agile team</td>
<td>Cross-functional teams</td>
</tr>
<tr>
<td></td>
<td>Team creates formal test report for release</td>
<td>[54,85]</td>
</tr>
<tr>
<td>C.2 FO over exposed to change requests</td>
<td>(Empower team to) update requirements based on learning from sprint</td>
<td>Cross-functional teams for requirements update [54,150]</td>
</tr>
<tr>
<td>C.3 Suffering requirements traceability</td>
<td>Explicitly define if user story must be traced</td>
<td>Increase understanding of process and roles [85]</td>
</tr>
<tr>
<td></td>
<td>Force team to understand reqts. model</td>
<td></td>
</tr>
<tr>
<td>C.4 All can write user stories anytime</td>
<td>Create proper channels for writing user stories</td>
<td>Cross-functional teams update requirements and peer-review [150]</td>
</tr>
<tr>
<td>C.5 Inconsistency between requirements model and user stories implementation</td>
<td>Have testers as part of team</td>
<td>Bring testers closer to requirement owner [151]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Iterative requirements management [152]</td>
</tr>
<tr>
<td>C.6 Complete picture and system thinking missing</td>
<td>Visualisation of how functions relate</td>
<td></td>
</tr>
</tbody>
</table>

### 4.5.3 Challenges Unique to Department B

**C.6: Complete picture and system thinking missing.** Traditionally, the teams had the full requirements and generally knew how they relate to each other. In agile development where the focus is on features, teams lose focus of how their functions relate to other functions. On the other hand, many different departments are using the same signal in different ways. One interviewee noted that some subsystems listen in to the signal without requesting for it. This makes it hard to know what effects a change in one unit could have on another unit.

*Proposed Solution:* While this is a classical problem in software development, it becomes more pressing in iterative and agile development at scale. All interviewees agreed that maybe having a complete picture may not be practical but recommended visualisation of how each and every function is related to each other. A good first step is to make explicit how each part fits into the larger picture. This can for example be achieved by adding relevance for tracing explicit for each backlog item (see C.3), or by making a formal test report part of hardening sprints before a release (C.1).


4.6 Discussion

Evidently, these two departments have adopted agile methods in different ways, and rely on different practices. Department B is more strict on user story management while Department A gives more freedom and responsibility to agile teams. Both departments embed agile teams in similar hierarchies defined from plan-driven systems engineering, but differences in the organizational interfaces affect the severity of requirements-related challenges faced. Department A teams have slightly more autonomy than teams in Department B and they frequently propose changes to FO. Figure 4.3 depicts the communication challenges introduced when departments define their own agile practices. Several studies \[40, 84, 153\] have mentioned and recommended organisations to tailor agility to their contexts. However, with agility being driven from the teams’ perspective, we find that the tailoring is more on the team level with different teams having varying flavours of agile and yet they still need to contribute to the same product. We thus also notice a challenge at the inter-team level even though we did not specifically explore that direction and leave it for future work.

Furthermore as shown in Figure 4.2 we noticed that Department A has one tool for requirements while Department B has several tools used by the different department roles. This diversity of tools makes it harder to see how the whole function operates or varies. Also, with many functions possibly connecting to other departments, the tools used in Department B make a difference in traceability and monitoring. Department A on the other hand seems to have less constraint on structures and thus face unique challenges which include: many user stories from different sources accessing the backlog and inconsistencies between the requirements model and implementation at testing time. These come up as consequence of less or not enough communication between testing and function owner team. On the other hand, Department B has teams that manage the requirements on a higher level and are thus able to monitor how the requirements change with respect to implementation. It is clear from the findings that the two departments have defined their own ways
of working, which contributes to the difference in severity of challenges faced in the departments.

In Table 4.2 we give an overview of how our results relate to related work. The cross functional teams have been proposed in literature to handle challenges C.1 and C.2. For this the teams would have to comprise of the required expertise to handle the update issues to FO and also manage high-level requirements within the team. For C.3 where it is not clear whose responsibility traceability is, Liebel et al. [85] recommend increased understanding of processes and roles to address the challenge of 'unclear responsibilities and borders'. It is worthy noting that these challenges could also be instigated by the failure to implement the agile practices well. Kuusinen et al. [40] recognise that as a challenge and recommend co-located teams and having a permanent workforce. In our opinion, this recommendation does not match our case context as the teams are already co-located and permanent to the extent to which it is possible to do so at this scale.

Though some potential solutions are recommended in literature, there is a lack of empirical investigation on their ability to mitigate requirements related challenges, as for example with respect to cross-functional teams.

4.7 Conclusion

In conclusion, we present results from the interplay of agile teams and traditional structures in software development. By studying the requirements flow with focus on the roles and structures used for requirements communication, we were able to identify challenges brought about by such coexistence. In both departments, and basing on the challenges observed, the current state of the coexistence of agile and plan-driven approaches seems to limit the efficiency of development. The departments have different ways of working and use different tools. Each team implements agile methods in their own way, which leads to some differences in the challenges we observed. Generally, all challenges relate to working with requirements updates when the team uses agile methods while the department structure is plan-driven. Strategies, in the literature, to mitigate these challenges range from having cross-functional teams, improving traceability through adequate tooling to improving process understanding and having well defined roles that may include empowering teams to manage requirements on their end. We believe that for agile automotive system development, a healthy way must be found that empowers teams to take more responsibility for high-level requirements, without sacrificing too much of the agile spirit. Practitioners in similar settings can use our findings to facilitate process improvement and to drive agile transformations. Future research will have to show to which extent these or other solutions can mitigate the challenges and to get most out of the combination of agile and plan-driven approaches.

While our challenges did not occur in the same way in pure plan-driven system engineering, they might not be specific to the interplay of agile and plan-driven. Future work will have to show if they also relate in pure large-scale agile approaches for system development.
Chapter 5

Paper D

Requirements Engineering Challenges and Practices in Large-Scale Agile System Development


Journal submission: In process of requested minor review
Abstract

Agile methods have become mainstream even in large-scale systems engineering companies that need to accommodate different development cycles of hardware and software. For such companies, requirements engineering is an essential activity that involves upfront and detailed analysis which can be at odds with agile development methods. This paper presents a multiple case study with 7 large-scale systems companies, reporting their challenges, together with best practices from industry. We also analyse literature about two popular large-scale agile frameworks, SAFe and LeSS, to derive potential solutions for the challenges. Our results are based on 20 qualitative interviews, 5 focus groups, and 8 cross-company workshops which we used to both collect and validate our results. We found 24 challenges which we grouped in 6 themes, then mapped these challenges to solutions from SAFe, LeSS, and our companies, when available. In this way, we contribute a comprehensive taxonomy of RE challenges in relation to large-scale agile system development, evaluate the degree to which they have been addressed, and outline research gaps.
5.1 Introduction

Despite wide criticism, agile approaches have significantly contributed to the way software is developed [30]. While initially focused on small teams [7,30,86,87], success stories have led to their application at large-scale [8,9,42] and in system development (i.e., large, complex systems which mix software and hardware) [9,10,43], an environment that is characterized by long lead times [10] and stable, sequential engineering practices [44]. These complex, agile environments often involve many challenges which fall under the umbrella of Requirements Engineering (RE), including understanding product value, communicating product purpose, dealing with cross-cutting concerns [138], and managing requirements [88]. Because of these and other challenges, companies struggle to implement efficient RE in a large-scale agile context [14,89,90].

Existing work looking at RE-related challenges arising from agile methods, i.e. agile RE (e.g., [19, 23, 54]), mostly focus on proposing new approaches, practices, and artifacts [16]. There is however a lack of empirical studies that investigate the phenomenon of RE in relation to agile methods, particularly in the domain of large-scale system development [15,16,23]. This gap is a major obstacle when transitioning to agile system development at scale, considering the extraordinary demands on long-term maintenance, synchronisation of different development cycles (e.g. between hardware, mechanics, and software), and often safety concerns of today’s systems. Therefore, in this work we report the RE-related challenges of large-scale agile system development and their solution candidates. We refer to large-scale agile system development as the development of a product consisting of software, hardware and potentially mechatronic components that includes more than 6 development teams [8] and is aligned with agile principles [30]. We find such situations frequently at our industrial partners.

Through a multiple case study of seven large-scale system development cases, based on 5 focus groups, 8 cross-company workshops and 20 semi-structured interviews, as well as a review of state-of-the-art large-scale agile frameworks, we answer the following three research questions from an RE perspective:

**RQ1:** How pervasive are agile methods in large-scale system development? Given that our case companies have diverse backgrounds, we describe for each case how agile they are, thus determining the pervasiveness of agile development for that case company. Our aim is to better understand the general context of potential agile- and RE-related challenges in industry.

**RQ2:** Which requirements-related challenges exist in large-scale agile system development? Building on RQ1 results, we categorize and describe the challenges provided by our case companies.

**RQ3:** Which approaches have been proposed in popular literature and which approaches are used by practitioners to address the challenges identified in RQ2? Using RQ2 results as a benchmark, we aim to provide a set of solutions from proposals presented by practitioners as well as those offered by well-known large-scale agile frameworks, particularly LeSS [154] and SAFe [155].

In answering our research questions, this paper makes three contributions. First, we present a report of industrial RE challenges related to applying agile development in large-scale systems. We are able to classify our found challenges in six general areas: 1) Build and maintain shared understanding of...
customer value 2) Support Change and Evolution 3) Build and maintain shared understanding about system 4) Representation of Requirements Knowledge 5) Process Aspects, and 6) Organizational Aspects.

Second, the paper provides candidate solutions to the challenges identified under each of the challenge areas. The solutions are obtained from the use of established large-scale agile frameworks and additional solutions from best practices in industry provided by our industry partners.

Finally, we are also highlighting the need for systematic approaches to engineering requirements, even in an agile context. Thus, we hope that our work helps to establish RE practices that better support agility within large-scale system development.

This paper is based on preliminary work presented by Kasauli et al. [138], which reports the RE challenges in our multiple-case study. Using data from two more six-month iterations with three additional companies, this work revises and extends Kasauli et al. [138] as follows:

- We refined and expanded the catalogue of challenges by talking to the initial four companies plus additional three companies with comparable context.

- We used the expanded catalogue to run a workshop on RE practices in large-scale agile system development where companies described their ways of working, relating to the challenges. This activity contributed new potential best practices.

- We analysed documentation of SAFe and LeSS to understand to what extent we can rely on these scaled frameworks for addressing our challenges. The analysis together with feedback from the workshop provided potential solutions to our identified challenges.

This paper is organized as follows. In Section 5.2, we discuss the background of large-scale agile development and the related works to our study. Section 5.3 presents the research methodology we used including data collection, analysis and validity threats. We report our study results to RQ1 in Section 5.4. In Section 5.5, we report the results to RQ2 and RQ3. We discuss our results in Section 5.6, before concluding our article in Section 5.7.

5.2 Background and Related Work

5.2.1 Large-Scale Agile

Agile methods like Scrum and XP are being adopted in large-scale system development companies [42], even though they were originally intended for use on a small scale [7,86,87]. Existing work on this topic shows that companies successfully adopt agile methods, but that several challenges remain. In a survey with 13 organizations in 8 European countries and 35 individual projects on the adoption of XP and Scrum, Salo and Abrahamsson [42] report successful adoption of these methods and appreciation among practitioners. Lindvall et al. [13] study the potential of adopting agile methods with ABB, DaimlerChrysler, Motorola, and Nokia. The authors’ conclusion is that, overall,
agile methods could suit the needs of large organizations, in particular for small and collocated teams. However, integrating agile into the company environment could be challenging. Lagerberg et al. report based on a survey at Ericsson that applying agile on a large scale facilitated knowledge sharing and effective coordination. Additionally, through a questionnaire based survey of 101 Norwegian projects, Jørgensen analyse agile methods’ use for large software projects and conclude that increased use of agile methods in large-scale projects reduces failure risk.

In a systematic literature review on the adoption of agile methods at scale, Dikert et al. identify 35 challenges, e.g., coordination in a multi-team environment with hierarchical management and organizational boundaries. In a structured literature review on challenges of large-scale agile, Uludag et al. identify 79 stakeholder specific challenges, e.g., coordinating multiple agile teams that work on the same product, which was deemed specific to program managers. In a position paper by Eklund et al., research challenges of scaling agile in embedded organizations are presented. These challenges include, e.g., coordination of work between agile teams or taking into account existing ways of working for systems engineering. Similarly, Berger and Eklund present, based on a survey with 46 participants, expected benefits and challenges of scaling agile in mechatronic organizations, including efficiently structuring the organization, understanding of agile along the value chain, and adaptation to frequent releases.

In an attempt to address these challenges, several companies are adopting large-scale agile frameworks such as Scaled Agile Framework (SAFe) and Large-Scale Scrum (LeSS). These frameworks offer a series of practices, principles, and methods for large-scale agility, e.g., sprint-review bazaars, enabler user stories, guilds and chapters. Given the attention that these large-scale agile frameworks currently receive, we aim in this paper to discuss RE-related challenges with the principles and practices suggested by SAFe and LeSS. We avoid giving a full summary of these complex frameworks here, but refer to and briefly describe various specific practices and principles which address our identified challenges.

5.2.2 RE and Agile

In the past, agile methods and requirements were often perceived as conflicting, particularly if RE is seen narrowly as a set of “the system shall...” statements, which agile’s de-emphasis on documentation recommends to avoid. However, RE is a wide field covering requirements of all formats, implicit or explicit, including sharing and coordination of functionality, quality, or value-related knowledge. Although there is less work on the relationship between agile and RE, compared to work focusing solely on agile, existing work has commented on synergies and conflicts of traditional RE thinking with agile methods.

Based on a mapping study with 28 analyzed articles, Heikkilä et al. find that there is no universal definition of agile RE. Furthermore, they report several problematic areas in agile RE such as the use of customer representatives, prioritization of requirements or growing technical debt. In a case study by the same authors at Ericsson, the flow of requirements in large-scale agile is studied. Perceived benefits include increased flexibility, increased planning
efficiency, and improved communication effectiveness. However, the authors also report problems such as overcommitment, organizing system-level work, and growing technical debt. Similarly, Bjarnason et al. [54] investigate the use of agile RE in a case study with nine practitioners at one large-scale company transitioning to agile. The authors report that agile methods can address some classical RE challenges, such as communication gaps, but cause new challenges, such as ensuring sufficient competence in cross-functional teams. In a case study with 16 US-based companies, Ramesh et al. [19] identify risks with the use of agile RE. These are, e.g., the neglect of non-functional requirements or customer inability. A systematic literature review on agile RE practices and challenges reports eight challenges posed by the use of agile RE [15], such as customer availability or minimal documentation. However, the authors also report 17 challenges from traditional RE that are overcome by the use of agile RE. The authors conclude that there is more empirical research needed on the topic of agile RE.

Other studies have addressed the use of traditional RE practices and agile RE. Paetsch [17] provide a comparison between traditional RE approaches and agile software development while identifying possible ways in which agile software development can benefit from RE methods. The authors conclude that agile methods and RE are pursuing similar goals in key areas like stakeholder involvement. The major difference is the emphasis on the amount of documentation needed in an effective project. Meyer, in contrast, regards the relationship between RE and agile more critical, describing the discouragement of upfront analysis and the focus on scenario based artifacts (i.e. user stories) as harmful [30], however not based on empirical data.

In summary, there is substantial existing work on the adoption of large-scale agile in system development, including empirical studies. However, existing work either focuses on identifying and evaluating agile RE practices [15][16], or at presenting the current state of practice at single companies [23] and without explicitly targeting system development [54]. Hence, additional empirical work is needed to understand the complex phenomenon of agile methods and RE in the domain of large-scale system development. Our study contributes with a cross-case analysis of large-scale agile development.

5.3 Research Methodology

We have conducted our multiple-case study [76] in two rounds with several points of elicitation and validation in each round. In Round 1, we investigated four companies with a series of focus groups and interviews, the results of which have been summarized in [138]. In Round 2, new to this paper, we continue to investigate the four original companies, and a further three companies, running a series of cross-company and individual company workshops - gathering challenges and solutions and presenting and validating our findings with the case companies. The final results provide insights regarding our three research questions.

5.3.1 Case Companies

Our study includes one telecommunications company (referred to as Telecom
in this paper, two automotive companies (Automotive 1 and 2), one company
developing software-intensive embedded systems (Technology 1), another tech-
nology and engineering company (Technology 2), one manufacturing company
(Manufacturing) and one processing company (Processing). Both Manufac-
turing and Processing have significant software components. All seven cases
represent large, international companies developing products and systems that
include a significant amount of software, hardware, and typically mechanical
components. All case companies have experience with agile software teams and
have the goal to further speed up the development of their software-intensive
systems. We elaborate on the specific cases with respect to RQ1 in Section 5.4.

Table 5.1: Data Sources First Round (reported in [138])

<table>
<thead>
<tr>
<th>Type</th>
<th>Company</th>
<th>Role(s)</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus Group</td>
<td>Telecom</td>
<td>2xTest Architect, System Manager</td>
<td>FG-1</td>
</tr>
<tr>
<td>Focus Group</td>
<td>Automotive 1</td>
<td>Process Manager, Specialist Platform Software</td>
<td>FG-2</td>
</tr>
<tr>
<td>Focus Group</td>
<td>Telecom</td>
<td>2xTest Architect, System Manager</td>
<td>FG-3</td>
</tr>
<tr>
<td>Focus Group</td>
<td>Automotive 2</td>
<td>System Responsible, 2x Function Owner, System Quality Engineer</td>
<td>FG-4</td>
</tr>
<tr>
<td>Focus Group</td>
<td>Technology 1</td>
<td>RE Change Agent, Chief Engineer</td>
<td>FG-5</td>
</tr>
<tr>
<td>WS</td>
<td>Automotive 1</td>
<td>Verification Manager, Specialist Platform Software</td>
<td>XComp 1</td>
</tr>
<tr>
<td>Telecom</td>
<td>Test Architect, System Manager</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WS</td>
<td>Automotive 1</td>
<td>Verification Manager, Specialist Platform Software</td>
<td>XComp 2</td>
</tr>
<tr>
<td>Telecom</td>
<td>Test Architect, System Manager</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Automotive 2</td>
<td>Telecom</td>
<td>Test Architect, System Manager</td>
<td></td>
</tr>
<tr>
<td>Automotive 1</td>
<td>Technology 1</td>
<td>Chief Engineer Software</td>
<td></td>
</tr>
<tr>
<td>Int</td>
<td>Telecom</td>
<td>Test Architect (TA), System Manager (SysM) x2, Developer (T-Dev), Scrum Master (ScrM), Area Product Owner (APO), Operational Product Owner (OPO),</td>
<td>T.*</td>
</tr>
<tr>
<td>Int</td>
<td>Automotive 1</td>
<td>Safety Technology Specialist (TS), A1-TS</td>
<td>A1-TS</td>
</tr>
<tr>
<td>Int</td>
<td>Automotive 2</td>
<td>Component Design Engineer (CDE), System Design Engineer (SDE), Function Owner x2 (FO), Software Developer x2 (SD), Product Owner (PO), Scrum Master (SM), System Tester (ST), Functional Tester (FT), Software Quality Expert (SQE)</td>
<td>A2-*</td>
</tr>
<tr>
<td>Int</td>
<td>Technology 1</td>
<td>Requirements responsible</td>
<td>Tec-SRR</td>
</tr>
</tbody>
</table>
Table 5.2: Data Sources Second Round

<table>
<thead>
<tr>
<th>Type</th>
<th>Company</th>
<th>Role(s)/Purpose</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS</td>
<td>All companies invited</td>
<td>Planning and Validation</td>
<td>XComp PV</td>
</tr>
<tr>
<td>WS</td>
<td>Telecom</td>
<td>Verification Manager, System Architect</td>
<td>TelWS</td>
</tr>
<tr>
<td>WS</td>
<td>Technology 1</td>
<td>Project Manager, Chief Engineer, Requirements Manager, Technical Integrator (2x), Product Manager</td>
<td>TechWS</td>
</tr>
<tr>
<td>WS</td>
<td>Automotive 2</td>
<td>Project manager (Process, methods, tools) 3x (OD, SW-Dev, Sys-Dev), technical expert</td>
<td>Auto1WS</td>
</tr>
<tr>
<td>WS</td>
<td>Manufacturing</td>
<td>Project Manager x2, Chief Engineer, Electronic Developer, Technical Integrator x2</td>
<td>ManWS</td>
</tr>
<tr>
<td>WS</td>
<td>Processing</td>
<td>System Engineer x3, Technology Specialist x2, Project Manager, Requirement Manager</td>
<td>ProcWS</td>
</tr>
<tr>
<td>WS</td>
<td>Telecom, Technology 1, Technology 2, Automotive 1, Automotive 2, Processing</td>
<td>Toolchain and Processes, Requirements Expert, Process/Method/Toos(SysEng) x4, Process/Methods/Tools (SW) SW Reqt Eng. x3, Architect x2, System Manager</td>
<td>XComp 3</td>
</tr>
<tr>
<td>WS</td>
<td>Telecom, Technology 1, Technology 2, Automotive 2</td>
<td>System Engineer x2, Agile Team Lead, Product Owner x2, Agile Expert x2, Requirements Engineer x3</td>
<td>XComp 4</td>
</tr>
</tbody>
</table>

5.3.2 Sampling and Data Collection

In order to answer our research questions, we collected data both from our company cases and from the literature concerning scaled agile frameworks. Generally, we relied on semi-structured interviews (one or more interviewers interact with one or more interviewees based on an interview guide), workshops (a group meets to jointly work on creating a defined result), and focus groups (for a given scope, representative stakeholders are invited to discuss current challenges and future opportunities). In order to coordinate the multiple-case study, we relied on special cross-company workshops (XComp WS) for scoping of work, validation of results, and planning of next steps across participating companies. Figure 5.1 gives an overview of our research design for the first round of elicitation and validation, as presented in [138].

The individual elicitation events for the first round are summarized in Table 5.1. Starting from a common case study design and common research questions, we conducted a cross-company scoping workshop (XComp 1 Scoping WS) to secure commitment from participating companies, align the goals of the
study and finalize the research design. We then scheduled individual scoping workshops (Scoping WS) with each company, except for Technology 1 which, despite genuine interest in the study, could not free up resources for this study at that time. During these scoping workshops, we selected with the help of our company contacts the most appropriate case in terms of availability and available experience on the topic, e.g., a specific product or component (partially) developed with the use of agile methods. These cases were selected to accommodate two aspects: variation to allow better generalization of results and convenience, since there was an interest to investigate the research questions in each particular case. This allowed us to cover a variety of perspectives during data collection, i.e., system overview, customer experience, development, integration, and testing. Our generic data collection instrument for Round 1 of our study can be found online\(^1\). Data collection was adjusted according to each individual case based on resource availability and commitment. For instance, the Telecom case relates to a large product development by many Scrum teams and we relied on a focus group followed by interviews (denoted Int in the figure) with a variety of roles (see Table 5.1). In contrast, the Automotive 1 case relates to one Scrum team and we chose a focus group with the entire team, complemented with an interview of a safety expert. Interviews lasted approximately one hour and followed a similar interview instrument for all companies with domain specific adjustments for each company. For focus groups and cross-company workshops we scheduled three hours.

In the second round of data collection, summarized in Fig. 5.2, with elicitation events listed in Table 5.2, we relied on workshops to expand and validate our data.

All workshops were again scheduled for three hours. We started Round 2 with a cross-company workshop, XComp PV (Planning and Validation), at which we presented and received feedback on our previous findings, making plans for further investigation with each company. We then conducted a number

\(^1\) http://www.cse.chalmers.se/~knauss/2020-AgileREChallenges/
of individual workshops with each company, following a standard workshop instrument. The general purpose of the instrument was to both orient the new companies to the project and to go through the RE-related challenges found in the previous round of the study. In order to make the workshops more concrete, we elicited RE-related artifacts (e.g., product backlogs, feature descriptions), and understood how each artifact would relate to the challenges we had discovered. We conducted five company workshops (TelWS, TechWS, Auto1WS, ManWS, ProcWS) using this instrument as a guide, confirming, expanding and collecting challenges and solutions. Each workshop lasted three hours and was hosted by the case company.

In order to extract potential solutions provided in literature for large-scale agile development, we selected two popular large-scale agile frameworks to include in our analysis: LeSS and SAFe. Two authors read both sources and independently created a matrix relating challenges found with our companies to potential solutions suggested by either source. Our matrices were discussed and merged, and the results are presented as part of our findings, addressing RQ3.

We then conducted two, full day cross-company workshops, XComp 3 and XComp 4, with a focus on finding and developing solutions and strategies to our discovered challenges. In XComp 3, we presented an overview of current findings, including updated challenges, collected solutions from the companies, and the challenge-solution matrix obtained from the analysis of scaled agile frameworks, then discussed companies’ RE practices via individual presentations and a world cafe focusing on selected issues. In XComp 4, we had another full day workshop focusing on solutions to the challenges we found. Depending on availability, not all case companies could send representatives to each workshop, but each workshop had at least four companies represented (see Fig. 5.2). In addition to these focused workshops, at the beginning and/or the end of each six-month period, we conducted a further cross-company workshop (XComp PV) to orient and update case companies on our progress and to align on

---


CHAPTER 5. PAPER D

plans for the coming six months. These workshops gave us an opportunity to continually present our findings, receiving feedback.

Not all researchers participated in all interviews, workshops and focus groups. In Round 1 we had one dedicated researcher present in all data collection events. In Round 2, we had at least two out of three principle investigators present in all workshops, after calibrating our efforts via our research instrument. We recorded interviews and focus groups where possible and had at least two researchers take notes otherwise. Collected data was verified at multiple points with case company representatives through follow-up XComp workshops.

5.3.3 Data Analysis

For data analysis, we relied on a thematic coding approach [78]. For each case in Round 1, at least two researchers familiarized themselves with the data and highlighted noteworthy statements and assigned a label or code to each. Based on a card sorting approach, the authors of [138] discussed and iteratively combined codes into 30 candidate themes, from which we derived four high-level clusters containing 3-5 themes each. In our expanded data collection, material collected from company workshops was processed in the same manner, with two researchers sorting and updating findings to create the updated list of issues and solutions presented in Section 5.5. In each round, to validate the clusters, we discussed the outcome of our analysis in a reporting workshop with all participating companies.

As an example of our coding process, in round 1, interview A2-PO said the following, “I don’t think traceability is not required or something like that. It’s just that my focus hasn’t been on documenting the function.” These and other quotes led to the creation of code (challenge) ‘C3.c Creating and Maintaining Traces’. In round 2, as part of our discussions with Manufacturing, we made the note “Reusable modules = requirements and solutions”, providing a potential solution for our earlier C3.c challenge. These and other items are described in Section 5.5.3.3. As a third example, our notes from the ProcWS included the following statement: “Product focus means little reuse of requirements. How to reuse existing requirements in a new product? Beyond copy and paste...”. This and other supporting quotes and notes caused us to create a new code (challenge), compared to the round 1 results in [138], ‘C2c) avoid re-specifying, encourage re-use’, and is integrated with the other explanatory text in Section 5.5.2.3.

Once the challenges were established, we repeated the analysis process going through material collected from company workshops and the LeSS [154] and SAFe [155] sources, looking for potential solutions which mapped to our identified challenges.

Two researchers did this mapping individually. The results were merged and differences discussed. Section 5.5 presents both the challenges, updated from [138], along with potential mapped solutions, extracted from the company interactions and SAFe and LeSS materials. This list of potential solutions, mapped to challenges, was also presented back to company participants in a cross-company workshop, collecting feedback and making updates to the findings.
5.3.4 Threats to Validity

By design, the external validity of case studies is low. Hence, generalization of our findings might not be possible to different companies or domains. In particular, we cannot reason about challenges for small-scale or pure software development. We believe that while some challenges might be visible there as well, they can likely be managed ad hoc or within the scope of agile practices. We designed our study to identify common challenges across participating companies. Thus, our research method does not support any deep argument about differences between companies, domains, and market positions. However, given that we found similar themes in all cases, we expect that these apply similarly to other companies or projects in large-scale systems engineering.

To increase internal validity, we discussed the results of our analysis in multiple cross-company workshops (XComp 2 Validation WS, XComp PV). The workshops included key roles from each company that were already involved in the study. We also used the workshops to discuss underlying root causes and challenges that are shared by all companies.

To avoid a too restricted view on smaller parts of a project or a product, we selected interviewees from different parts of the development, including at least one team and several system level roles in each case. Workshops often involved a variety of roles from a variety of divisions/areas within the companies. We relied on a convenience sample and companies provided us with access to dedicated company experts in the areas of agile transformation and RE, with a genuine but diverse interest in the field. While we hope that this improved internal validity, it might have introduced a selection bias, which we tried to mitigate by encouraging participation of both proponents and opponents of agile/RE. Our contact persons at the case companies all have substantial knowledge in the area of agile transformation. Therefore, we expect that they were able to select suitable participants.

To mitigate threats to construct validity, we selected (through our company contacts) participants that have knowledge agile transformation and RE. Additionally, we designed and improved the interview guides in multiple iterations and with correspondence from the company contacts. During the interviews, workshops and focus groups, we gave explanations where concepts were not clear and asked participants or interviewees for elaborations in case of an ambiguous answer. Data was collected from multiple sources including different companies and existing literature (SAFe and LeSS) which helped us ensure we identified the challenges correctly. During data analysis, we used data triangulation between interviews, company workshops, and the literature. Further, in case of ambiguous statements, we would contact the interviewee or include a discussion of these statements in the next XComp workshop.

Reliability is hard to achieve in qualitative studies. However, we tried to describe our study design, in particular the data collection and analysis procedures, in a detailed fashion and shared the various instruments used for data collection. At least two researchers were involved in all interviews, focus groups, and workshops, to reduce the impact of subjectivity. Similarly, we analysed all data involving at least two researchers at a time. With all case companies, we have a prolonged involvement leading to mutual trust among the parties.
The potential solutions proposed in Section 5.5 are based on our own reasoning, claims in related work (that these solutions help with a certain challenge), and on discussions with the case companies. Thus far, we have not applied the solutions from the literature in the case companies, or solutions suggested by one company in further companies. Further validation of the collected solutions is needed.

5.4 Pervasiveness of Agile development (RQ1)

One of the first challenges we encountered when planning this study relates to characterizing the state of agility in potential case companies. Especially at scale, it becomes very hard to give an overview of concrete practices being applied within a development organization. As one of our company contacts stated:

“I suspect that our organization is very agile when judged on what is found on powerpoint level, but hardly agile at all when judged on how agile practices are implemented.” — Anonymous interviewee

The challenges we discuss in this paper may offer potential explanations for why adopting satisfactory agile practices is so hard. They should not be seen as challenges that only occur once a company has completely transitioned to agile, but more generally as challenges that companies need to consider when they aim to be agile.

In addition to the practices being followed, we found it important to distinguish how widespread agile approaches are in the company. Figure 5.3 is a simplified visualization of the different states we found within the case companies. For some, agile practices are only applied by software teams, for others, they span the development of complete functions (incl. hardware and potentially mechanics), and for some, the full development organization aims for continuous and agile development. Note that not only do the companies differ in the way they (aim to) implement agile practices, but that there is also a huge variation within the individual companies’ products, services and structures. This section provides an overview of the contexts of our case companies.

Telecom Company  The Telecom Case relates to the development of one major product. More than 30 Scrum teams develop in parallel based on a scaled agile approach (adapted from SAFe [82]). Scrum sprints are based on a backlog and a hierarchy of product owners breaks down product requirements and customer visible features to backlog items. While these product owners represent the customer requirements towards the product development, system managers represent a system requirements perspective. The overall effect is a continuous development stream and feature flow, which is supported by a powerful infrastructure that enables continuous integration and testing. Pre-development generates knowledge about new features, which enables effective planning for continuous delivery.

Particular to the Telecom case, hardware development is largely decoupled from the software development. New hardware becomes available with a regular, but low frequency. Thus, the software development sets the pace of system development, which can be seen as continuous and agile, in that it embraces
5.4. PERVERSIVENESS OF AGILE DEVELOPMENT (RQ1)

Figure 5.3: Different scopes of agile development within system development. (Light grey box denotes RE, arrows indicate how requirements are used for informing developers about what to implement, testers about what to test, and for documenting the system for maintenance. Three black boxes show the different agile scopes discussed in this case study.)

agile values as much as possible. In Figure 5.3 this is shown by the largest box, which implies that the whole scope of a traditional V model is covered.

**Automotive Company 1** In Automotive 1, agile methods have been successfully applied to in-house development of software components. In the light of growing competition from software-centric companies, e.g., on autonomous driving, there is a desire to scale up these fast-paced approaches from developing software components to developing complete functions, thus including agile development of hardware and mechatronic. The selected case is a pilot project that re-implements a whole customer function in an agile way. Integration of this function into a real vehicle requires additional verification with respect to safety and overall system behavior. Thus, we would characterize this situation with the second largest box in Figure 5.3, where a function owner takes responsibility for one particular function and implements it with an agile team.

**Automotive Company 2** With Automotive 2, we selected a case responsible for safety critical functionality developed in house. As with Automotive 1, agile teams develop software within a development process that still corresponds to the V model. Within the agile software teams, software requirements are transformed into backlog items. In order to speed up development of this differentiating functionality, different measures have been taken to speed up the overall system development, such as introducing a shared information model that supports storing requirements, design elements, tests, and implementation models throughout the system development. Since this helps shortening development time significantly, participants referred to this approach as narrow V model (comparable to agile loop in [43]) in FG-4. In Figure 5.3 we describe this as the smallest box, not to refer to overall development speed, but to the fact that hand-over between plan-driven and agile development happens on a low level of abstraction.
Technology Company 1  Technology 1 develops mechatronic products, both for consumer markets and for industrial development and manufacturing, as well as for OEM system integrators. Their system development is decomposed into several system elements. Software development is mostly confined to two of these elements, both of which are characterized by agile methods and practices such as Scrum and Continuous Integration. As with Automotive 2, we refer to this situation with the smallest box in Figure 5.3, as Technology 1 enables agile work of more than 20 Scrum teams within a plan-driven system development organization.

Technology Company 2  Technology 2 develops advanced systems both for consumer markets and for OEMs, including systems that are safety critical. In order to better serve their customers, Technology Company 2 is increasing their agile development competency. Especially for interacting with OEMs, this entails challenging established ways of working throughout the company, and at the time of this investigation, we discussed agility on the scale of a full customer project.

Manufacturing Company  The manufacturing company develops high-tech products of supreme complexity and very large software parts for the medical domain. In order to decrease lead-time for delivering new features and to increase throughput, continuous development paradigms have been embraced throughout the R&D department. Agile principles and practices are considered on all levels, yet must be carefully considered due to regulatory requirements and the very large scale of the development effort. The software development is to a good extent independent from hardware development cycles and can be considered very large scale. It is increasingly organized according to large-scale agile development frameworks and continuous software development paradigms and at this scale, we would characterize its level of agility to correspond with the second largest box in Fig. 5.3.

Processing Company  The processing company offers components, services, and management for production and factories. Services provide detailed intelligence about physical processes within a factory or plant. The company has adopted agile ways of implementing software based services, which however rely on capabilities of physical components within a factory or plant. Thus, the scope of agility roughly relates to the smaller box of Figure 5.3, when considering a complete facility as the system.

Summarizing the seven cases, we recognize that some case companies have come a long way towards continuous software engineering and enterprise-wide adoption of agile [91]. Others are currently moving in that direction. Our research aims for common themes, regardless of the pervasiveness of agile adoption (for which we control with RQ1) or agile maturity (which we did not explicitly investigate in this study). In the analysis of interview and workshop data, we uncovered challenges and practices that relate to the application of agile methods in these contexts, described in the next section.
5.5. CHALLENGES AND POTENTIAL SOLUTIONS (RQ2 AND RQ3)

With respect to RQ2, we see 25 challenges that we group into six areas of challenges: Build and Maintain Shared Understanding of Customer Value, Support Change and Evolution, Build and Maintain Shared Understanding about System, Representation for Requirements Knowledge, Process Aspects and Organizational Aspects. Although we have tried to organize the challenges in similar areas to facilitate understanding, the areas are not independent, and often there is overlap between areas and challenges. See Fig. 5.4 for an overview. In the rest of this section we present the challenges along with their solutions candidates from SAFe and LESS, and, when available, solutions suggested by our participating companies.

### 5.5.1 Build and Maintain Shared Understanding of Customer Value

#### C1.a: Bridge gap to customer

C1.a: In large organizations, it is challenging to achieve sufficient customer collaboration. It is hard to make teams understand customer value, express actual customer value in terms of user stories that can be implemented in a single sprint, as well as to provide feedback to and obtain clarifications from the customer.

Despite the close customer relations advocated by agile, study participants indicate a large distance between customers and developers. In all our cases we found dedicated roles that channel information from multiple stakeholders down to the teams. It is not trivial to bridge that gap, direct interaction of teams and stakeholders can lead to chaos when established plans are circumvented or on-site customers are not an option. Teams want to be agile, but do not focus on customer value.
**i. Make team understand customer value**  Independent of distance/gap, the teams struggle to understand their customers’ view and cannot describe how their work provides customer value. Teams work with sub-features and tasks that can be finished during a typical sprint as opposed to the bigger features in order to ensure frequent delivery, a practice noted from all our case companies, although the methods used differ. However, one interviewee (T-ScrM) pointed out that a *feature is what is sold to the customer*. It thus becomes hard to gauge what the value of a sub-feature is. One participant (XComp 1) claimed that the focus on agile practices occupied the teams so much that this caused a neglect of product value. *Teams just want to be agile.* However, value creation is not solely the teams’ responsibility as the requirements breakdown starts from the customer units, as in the Telecom case, or from the function management units, in the Automotive cases. One interviewee (T-APO-1) pointed out that it is hard to break down the requirements such that they carry user value, a challenge also recognized in other cases (Automotive 1 and Technology 1).

**ii. Unable to express value in user stories**  Due to complexity of systems it is hard to write user stories that can be addressed by one team in one sprint and at the same time relate to value that could be recognized by a user/customer. User stories provide a fast means to share knowledge both on a high and a low level in an agile system development. In the Telecom as well as in the two Automotive cases, user stories are used for two purposes:

> “... so there are user stories that of course take the view from the end customer and describe what the end customer wants from our system and why. But then there are other user stories that are more like work descriptions of what the team should achieve and those could be like internal things that need to be developed in order to keep the architecture constrained.” — T-SysM

A Function Owner in Automotive 2 specifically expressed that high-level user stories could help to communicate value early. However, it is particularly difficult to write user stories that have direct value for the user. Such user stories would typically be too large to be completed and demonstrated in one sprint. Yet, breaking it into more user stories or more detailed requirements could deteriorate requirements quality since not enough effort goes into maintaining the requirements. This also creates traceability challenges, as it is hard to understand which high-level user story can be traced to detailed requirements. We discuss traceability further in challenge C3.c. In summary, user stories are hard to write at the scale and complexity of the cases in our study, yet they offer a unique opportunity to bridge distances between customer and developer.

**iii. Feedback and clarification**  Our teams suffer from long feedback cycles, which are a consequence of a) dependence on slow hardware development/deployment, b) customers not being agile, c) large numbers of stakeholders. In several companies, study participants raised the issue of long or complicated feedback cycles. At Automotive 1, one study participant named slow mechanical or hardware development as one of the main reasons for long feedback cycles. If software has to be tested together with actual hardware, feedback on software functionality is postponed until the hardware is ready. One study participant stated a second reason — often customers are not agile and take a long time to try out and approve new features. By the time feedback
then reaches the agile teams, they are already working on another part of the product and do not remember exactly what the feedback is about. That is, for the teams the feedback comes too late, while customers do not see value in giving quick and frequent feedback, even on smaller increments. This challenge is especially encountered if the system under development is supposed to be integrated into a larger system at the customer site, as for example in the Telecom and Technology cases. A third reason for complicated feedback cycles is that there is a large number of stakeholders, both external and internal. Due to the complex nature and the scale of the products developed by our case companies, there is rarely a single customer. Instead, requirements inflow occurs from many different sources, e.g., customers, authorities, managing subcontractors and sourcing, or standardization organizations. In many cases, requirements need to be discussed with and communicated to other stakeholders within or outside the organization, delaying feedback.

Potential Solution  In our view, the root cause of these challenges relates to size and complexity of the systems we investigated. In such large-scale systems, customers and end-users cannot easily relate or give feedback on things developers work on.

In our workshops and focus groups, participating companies brought up potential solutions. Many of these relate to facilitating discussion and communication, which could for example be supported through better visualization of requirements (Technology 1) or by investing into teams to focus on customer value every sprint (Telecom). Further, the companies suggested that handovers should be reduced, e.g., by introducing cross-functional teams that span traditional levels of abstraction (FG-1 and FG-5) or by keeping the product management close to development teams (ProcWS).

Common frameworks for large-scale agile, such as SAFe [155] and LeSS [154], focus on customer-value and offer advice that relates to our challenges. SAFe generally suggests frequent (train) demos as well as to involve customers on every level [155]. Similarly, LeSS recommends organizations to be customer-centric [154]. While we agree with this advice, we suggest that more concrete support must be provided in the light of our challenges. If work provided by an individual team does not clearly relate to a feature for which a customer could care, it will be hard to demo or to involve customers in decisions.

In line with Lean Software Development, SAFe also suggests that teams take an economic view, which, if sufficiently supported within an organization can help [154,155]. In addition, SAFe suggests that teams should cover all necessary roles [155], which might help, but could also be problematic, since the inter-disciplinary nature of large-scale system development may lead to a large number of necessary roles. LeSS offers Sprint-review bazars [154], which might offer teams an opportunity to practice relating their work to customer-value. Other than that, LeSS suggests to rely on product owners to connect teams and customers [154], but does not share concrete advice or tools for product owners to navigate the challenges we bring up.
5.5.1.2 C1.b: Building long-lasting customer knowledge

C1.b: In complex product families and large stakeholder landscapes, it is hard to maintain reusable knowledge about customers. Thus, each change could result in repeated efforts to acquire similar information from customers.

Even if the challenges related to feedback and clarification can be addressed, gained knowledge must be effectively managed, as pointed out by participants in FG-5 and FG-3.

“The teams have a lot of tacit knowledge, which is not available beyond their scope. But how much ceremony should we force on teams?” — FG 3

Even beyond designing a single system, knowledge about customers and their needs should be maintained for future projects. Without a good knowledge management approach, this can collide with the desire to allow empowered component teams to make fast, local decisions. Two aspects of this knowledge management challenge were raised: First, it is unclear where knowledge about a specific customer can be managed beyond the team and current project. Second, in continuous product development, teams might not realize that they have valuable knowledge for other parts of the system development, while those other parts do not know that valuable knowledge is available.

Potential solution  Both, SAFe and LeSS focus on short lead-times. It appears that long-term knowledge is mainly captured as (automated) tests and in the product itself, but also maintained within the agile organization. To this end, we find discussions about component and feature teams within the SAFe community insightful. In particular, the community indicates a slight preference towards feature teams [157].

In contrast, component teams can maintain long-term knowledge about which features their component supports and how this is providing customer value. However, problem-based customer or end-user requirements must be translated into requirements that a particular component should fulfill. This additional indirection is likely to increase the team-customer gap.

We did not find relevant practices specifically for building long-lasting customer knowledge in LeSS. Our participating companies could not provide further potential solutions beyond the use of feature or component teams.

5.5.2 Support Change and Evolution

5.5.2.1 C2.a: Managing experimental requirements

C2.a: When exploring new functionality or product ideas, experimental requirements need to be treated differently from stable requirements. Still, they need to be captured and potentially integrated in the system view at a later time.

Organizations that develop large, complex products have often established significant research and pre-development operations as part of traditional
systems engineering. When changing to an agile organization of system development, it is not clear where such activities (which can easily span a year) fit in. Should a particular cross-functional team research, create a prototype, and then develop a specific system function? This does not likely fit well into an agile iteration and release rhythm. Should a specialized market/research department do such activities? This would introduce hand-overs, often including comprehensive documentation, which would appear non-agile.

On the scale of typical system functions or user-visible features, research and pre-development also asks for explicit support for managing experimental requirements. That is, given a current state of the system requirements, it should be able to create a variant, exploring what-if scenarios and identifying potential changes to the overall requirements model that a specific new function or feature might entail. Given the scale of products and their features, it is clear that the current state of the system requirements will evolve during such research and pre-development activities. Thus, our case companies were raising the need to create, synchronize and merge variants of the system requirements.

**Potential Solution** In our workshops, participants were considering to introduce specific sprints for learning and increment planning. It seems generally more promising to broaden the views of team-members and allow them to participate in such activities, and by this to reduce hand-overs. With respect to the actual managing of variants, there was a suggestion to manage requirements as part of the product. T-Reqs, a specific solution that we explored, considers to maintain (textual) system requirement in the same repository as tests and source code [150]. This allows to rely on powerful support for branching and merging that modern source control systems such as git provide.

SAFe proposes some mechanisms that can support the management of experimental requirements, e.g. by relying on *enabler stories, architecture, and exploration* [155, p. 108]. Further, *set-based design* allows to some extent to reason about different alternatives during the development flow [155, p. 178]. Input from exploration, research, and pre-development can also be managed as *(variable) solution intents* [155, p. 186ff].

LeSS, in contrast, appears to suggest that this complex topic can be handled using a backlog [154].

In summary, we identify encouraging building blocks for solving the challenge of managing experimental requirements, but have to note that combining them into a convincing strategy remains non-trivial.

### 5.5.2.2 C2.b: Synchronization of development

C2.b: In large organizations, there exists a large variety of stakeholders, teams, projects and features. This variety makes it challenging to synchronize development between teams. A trade-off arises between documenting extensively and specializing teams to take ownership of a single feature or system aspect.

In many of our case companies, teams receive requirements from the product managers through several organizational levels. Furthermore, they often need
to exchange information with other teams to synchronize the development. This process of channeling the ‘right’ information towards and between teams is difficult and time-consuming. Hence, it limits agility and speed of teams.

FG-2 participants wondered whether agile should be limited to the development only, or should start from a feature request. In the former case, developers would receive feature requests in the form of already broken down requirements for implementation. In the latter case, developers would have to do the breakdown of a feature request into smaller units themselves. While both cases seem to be feasible, the question is how teams can be synchronized in any of these cases, especially at scale, where some form of decomposition is required. If requirements are broken down by an external role or team, possibly in a plan-driven way, they can be handed to different agile teams and their work needs to be synchronized. If they are broken down and implemented within one team, multiple agile teams only need to synchronize when there is interaction with or dependencies to features developed by other teams. However, analysis of a user-visible feature with respect to its implications and suitable decomposition takes time and it is not clear how this work can be fit into the tight sprint schedule of agile teams. Awareness about such dependencies is a pre-requisite.

**Potential Solution**  As described above, our company partners suggest different levels of agile pervasiveness as a way to address this issue, although different choices have different tradeoffs. SAFe suggests to provide such synchronization through bi-weekly synchronisation of agile release trains (ART, a set of agile teams that work together towards a shared release schedule) [155]. This is further supported through enabler stories for exploration [155, p.108].

With respect to organizing such synchronization, SAFe also suggests tribes (i.e. organizational units of around 100 members in a common scope, such as an ART), chapters (i.e. communities of practices, that can discuss cross-cutting concerns within a tribe), and guilds (allowing to discuss cross-cutting concerns beyond the scope of a single ART or tribe) [158, p.46]. We believe that such structures provide good support for synchronization of development, mainly however for discussing methods and processes. It remains an open question whether for example a safety or performance guild could also provide value to discuss cross-cutting requirements. Further, the *unity hour*, a regular meeting meant to bring together a tribe [158, p.33], can be used to make announcements that can foster synchronization between teams.

LeSS is comparably brief on the synchronization of development, but suggests aiming for continuous improvement based on reflection, both on team level and overall [154, p.69].

### 5.5.2.3 C2.c: Avoid re-specifying, encourage re-use

C2.c: Focusing on projects discourages re-use between projects. Defining a strategy to manage existing requirements and encourage their re-use across projects is challenging.

Our company partners have indicated that dealing effectively with legacy
systems is becoming more important. Previous approaches which focused on projects instead of products or components lead to re-inventing common requirements. Several of our companies are searching for ways to reuse existing requirements, e.g., beyond copy and paste (Processing). One participant company with a shared requirements database indicated that reusing requirements was still a challenge, as reusing requires a general knowledge of existing features – in order to reuse, one needs to know what is there. Currently, requirements reuse only happens on the lowest levels.

Requirements reuse also potentially involves some level of governance. On one hand, it may be desirable to avoid duplicate or very similar requirements in the repository (e.g., for different projects or products), but on the other hand agile teams want freedom and autonomy in their practices.

**Potential solution** In our workshops, we explored two different approaches to facilitate re-use of requirements: product line engineering and shifting from a project focus to a product focus. While the former approach would aim to group requirements into customer-visible and reuse-oriented features directly linked to existing solutions and components, the latter would bundle key requirements into a core product that can be maintained over longer time.

It appears that both SAFe and LeSS are assuming a product-focused organization. In such a context, *set-based design* could facilitate reuse [155, p.75,190]. LeSS suggests to avoid duplicate product functionality as well as a narrow product definition, which could to some extent remove the need to reuse requirements [154, p.159].

Despite these recommendations, SAFe and LeSS do not go into detail or emphasize systematic reuse in large-scale agile projects.

### 5.5.2.4 C2.d: Updating requirements

C2.d: Requirements can be defined at the beginning of the sprint, but often these requirements become out of date, and no longer reflect the solution. This causes issues in organizational memory. It is challenging to understand when and who should update requirements.

In TechWS, the company explained that their requirements and feature models are often old and not up to date. Requirements are not being updated, in part due to the nature of agile work, which does not explicitly factor in activities for keeping requirements up-to-date. This issue was echoed by Telecom and Manufacturing, in the latter case requirements are defined at the beginning of the sprint, but then kept the same; however, the company would like to enable more flexible requirements updates. In the Telecom case, non-functional requirements are kept in a document originating from before the agile transformation took place and for which there is no obvious way of providing regular updates. In the short term, this works, as these requirements are relatively stable; however, it is not clear within their current processes how to deal with this document becoming slowly out of date. From an agile perspective, perhaps the requirements only serve to get the development started, and it is therefore not important to keep them up to date with the eventual
product. However, the overall problem is that having out-of-date requirements can cause confusion, discourage requirements reuse, and prevents companies from using requirements as a form organizational memory, a practice which we found desirable in our subject companies.

**Potential Solution** LeSS acknowledges that requirements areas have a lifecycle in which they will change, get less important, or are retired [154, p.105]. This clearly shows that the problem is known, yet there is a lack of concrete guidance on how to do this. We believe that guilds and chapters in SAFe could be useful for bringing together interested parties in a platform that could make decisions about updates of cross-cutting requirements [158, p.46].

Still, we are surprised about the lack of explicit mechanisms for updating or changing requirements in large-scale agile frameworks. It suggests that epics and user stories do not provide value beyond planning the next releases and that updates from the agile teams (such as hidden dependencies or costs) are irrelevant for updating such planning as well as that no other requirements-related information (beyond for example tests) should be shared between teams. Neither of these suggestions match our data.

One suggestion from our focus groups is again based on T-Reqs, a system that allows cross-functional teams to manage system requirements together with source code and tests in a version control system [150]. T-Reqs allows all teams to update requirements via git and relies on gerrit for peer-reviews, which not only allows to check a team’s suggestion for updating or deprecating requirements, but also to share information about such updates with peers. Thus, requirements could be updated or deprecated based on knowledge generated during agile sprints. Note the strong relationship to our Challenge 4.C Tooling not fit for purpose (Sect. 5.5.4.3).

### 5.5.3 Build and Maintain Shared Understanding about System

While the C1 challenges focus on building and maintaining shared knowledge of the customer value, these challenges focus on building and maintaining knowledge about the system, a more internal view.

#### 5.5.3.1 C3.a: Documentation to complement tests and stories

C3.a: For complex systems, user stories and test cases are often insufficient to understand the overall functionality. It is challenging to complement these artifacts with appropriate but yet agile documentation of requirements that provides this understanding.

The idea of using test cases both as actual test artifacts and as requirements documentation is wide-spread in the agile community [30] and was also discussed by several participants. While this was seen as a potential way to reduce documentation effort, several issues with this approach were brought up. According to several study participants, test cases do not carry enough information to serve as a means of documentation:
“Tests are written in a pragmatic way. They do not capture the 'why.'” — Tec-SRR

Other interviewees throughout the companies added that one would need a number of tests to document any significant requirement, which will then be hard to reconstruct from just reading the tests during maintenance.

Several participants saw similar problems with user stories, as they would only reflect single scenarios. The overall system behavior would then emerge from the synthesis of all these single scenarios. To derive this full picture from tests or user stories only would, however, be too difficult:

“If we don’t specify this kind of complete [requirements] specification, we could try to use all [...] user stories [...]. But then we must base the understanding on [...] lets say [...] 2000 user stories [...] and try to find a good way of describing the complete system.” — T-SysM

It is interesting to note that this challenge surfaces early on, i.e., when an incoming customer request is analyzed. Therefore, if agile teams only develop backlog items based on finished requirements that they receive from other parts of the organization, they might not be aware of this challenge and therefore wrongly consider user stories complemented by test cases to be sufficient.

While in the Telecom case the issue of understanding system behavior from user stories or tests was mainly discussed with respect to new features, participants in Automotive 1 raised this issue especially for system maintenance. FG-2 participants agreed that user stories or test cases would not be appropriate to understand the behavior. They were unsure what form of documentation should be used instead, which level of detail the requirements should be on, and how they could be different from ‘traditional’ requirements.

**Potential Solutions**  With respect to complementing user stories and tests, the focus groups yielded suggestions relating to two areas: modeling customer-value (see C1.a and C1.b) and modeling (distributed) system behaviour. With respect to the latter, the reasoning is that teams could use models to explicitly describe the intended behaviour of their solutions. However, some of the same challenges with traditional large-scale requirements documents could arise with models, e.g., scalability, and modifiability.

Furthermore, we note that several companies create additional custom requirements documentation to supplement tests and stories. For example, Telecom creates a one-slide description of a feature, giving a high-level view, and then traces this description down to requirements.

SAFe provides suggestions to model both interfaces and behaviours to account for their importance. There have also been further suggestions in the large-scale agile literature to use models to analyze requirements [157, pg. 356], however, it is not exactly clear how these models will relate to the requirements information model suggested in SAFe.

We did not identify artifacts to potentially complement tests and user stories in LeSS. In contrast, the focus on specification by example and acceptance test driven development seems to suggest that such a complement is not anticipated in LeSS.

To summarize, modeling in an agile manner is one possibility to supplement the description of overall system functionality provided by user stories and test cases. Other forms of lightweight textual summaries can also be possible.
5.5.3.2 C3.b: System vs component thinking

C3.b: It is hard to balance system versus component versus feature thinking in complex system development with multiple teams.

Teams typically have specialized knowledge for their scope. However, they may lack the overall system knowledge. This can be problematic: when developing a feature in a complex product, as several components might be affected. Our companies had different types of agile software development teams. Some companies relied mainly on component teams, who become experts for their component, but do not necessarily understand all features that are supported by their component or the implications of their design decisions on the overall system. Thus, when reasoning about the quality of a component, teams might sub-optimize with regard to the overall system. Other companies relied mainly on feature teams, which might find an elegant way of implementing a new feature. Such feature teams struggle however to monitor the evolution of all affected components as well as their quality. Often, we even find a mixture of feature and component teams in complex systems, where some of the more sophisticated components are maintained and developed by dedicated teams.

Overall, we find it is challenging to provide teams with system-level knowledge while at the same time maintaining specialized knowledge about features or components in the teams.

Potential Solution In our focus groups, Automotive 2 clearly sees the need for a global baseline that allows to reason about the full system. In large system development, requirements can be seen as a way to put tasks on specific sub-organizations. Automotive 2 reported that this is not beneficial. Instead, requirements should be split with respect to the product while all parts of the organization should be encouraged to also monitor requirements for the full system, not only for their component. Processing further suggested that these ideas should be complemented by a clear model of ownership of requirements on all levels.

System thinking on all levels could, according to our focus groups, be facilitated by placing architects in teams and to take special care with respect to the architectural runway, when planning architectural enablers. Our companies saw the need for active governance of APIs, dependencies, and interfaces between teams, and to manage volatile architectural concepts differently from those that are stable.

It is one strength of SAFe to provide a clear breakdown hierarchy from enterprise level to teams [155], which, when combined with awareness on how each part fits in the complete picture, could discourage localized thinking. SAFe does promote system thinking [155] pg. 70f, [154] pg. 12], yet it is hard to deduct from the textbooks on how such thinking will emerge.

It is interesting to look at the discussion of feature or component teams [157], particularly from the perspective of this challenge (systems vs. component-thinking). Clearly, a feature team will find it easier to think about the system and how it relates to a particular feature. Yet, their goal will be primarily to implement the feature. The long-term quality of the different components as
well as their role for the overall system is not their main concern. One of the more concrete suggestions with respect to SAFe is to strive for *tribal unity* and use regular release train level meetings for communicating the vision [158, pg. 78,83].

Similarly, also LeSS emphasizes the importance of whole product focus [154, pg. 11,78], which is partially provided by multi-team product backlog refinement and the engagement of the product owner with the team to facilitate ownership of the product.

### 5.5.3.3 C3.c: Creating and Maintaining Traces

C3.c: Traces are valuable and often required, but rarely provide a direct value to their creators. Thus, they are typically produced inefficiently post-development and not maintained. It is challenging to incentivize the creation and maintenance of trace links.

In several of our companies, we see the existence of both textual requirements and user stories, where requirements are produced in a plan-driven way and provided to teams or organizations, who then create user stories to work locally in an agile way. However, since user stories relate directly to feature implementation they are not always systematically derived from existing requirements. Thus, direct tracing is not always possible.

A similar situation occurs in Automotive 2, where product owners write user stories based on plan-driven requirements they receive as an input. These user stories can in fact be rather local development tasks and backlog items that do not require tracing to system requirements. Thus, traces are not systematically managed, which can lead to additional work in cases where such backlog items become relevant for tracing to system requirements. The fact that often only the product owner is aware of which user stories originate from which requirements can slow down collaboration between agile teams and plan-driven RE teams. Interviewees in the agile teams considered tracing user stories to requirements to be documentation, which should not be part of the agile process. Instead they preferred to spend their time on implementation:

> “I don’t think traceability is not required or something like that. It’s just that my focus hasn’t been on documenting the function. I just focus on doing implementation and developing the function.” — A2-PO

This view was also shared in Automotive 1: while participants stated that tracing is valuable, or even required by standards, they felt that right now there is not enough incentive for agile developers to create traces. They wished for an incentive or directly visible benefit for the developers as well as for simplifying trace creation.

**Potential Solution** In our focus groups, most focus was on how to reduce the workload related to tracing. One way of doing this could be through a better approach to reuse, where complete features (equal to a group of requirements) are seen as reusable modules, consisting both of reusable requirements (with adequate tracing) and reusable solutions. By moving from project to (long-term) product focus, such reuse could further be facilitated, and agile teams would find themselves integrating existing requirements with strong traceability.
For system engineering companies, it is slightly concerning how little scaled agile frameworks discuss tracing. SAFe suggests to describe the solution, which will likely result in technical documentation with rich trace links [155, pg. 184ff]. LeSS suggests to link to wiki pages for additional information [154, pg. 33], as well as to suggest to link backlog items to ancestors for maximum three levels [154, pg. 204,222], but does not offer rich details on how and by whom such links are maintained.

5.5.3.4 C3.d: Learning and long-term knowledge

C3.d: Due to their long lifetime, product families require knowledge to be built up and maintained over longer periods of time and across products. It is challenging to optimize an organization towards generating and maintaining this knowledge, both on system level and on team level.

In the Tech1WS, the company expressed that agile is needed at the beginning of development, but later the need shifts to knowledge management, in part to support the learning of future personnel and other teams. Often requirements management is an activity that is performed at the end of the sprint. Technology 1, Manufacturing, and Processing expressed that spreading knowledge and networking knowledge was a challenge, it is not clear for them how to synchronize knowledge across different teams working in different cycles or on different projects. In the ManWS, the participants discussed the use of specialists to share knowledge (e.g., a specialist in security), but decided that this was not the most effective solution in practice, as often the specialists ended up being rare, and working in too many different contexts.

TelWS brought up the challenging trade-off between feature ownership and documentation. If a feature is strongly owned by an individual or team, it does not need extensive documentation; however, others are then dependent on the team or product owner for anything to do with that feature, and this places the company in a dangerous position in the case of personnel turnover, where important, non-documented knowledge is lost.

Potential Solution Although turnover is a problem, one solution offered by Telecom is to exploit ownership of features and tools to document less. One way to mitigate personnel loss would be to support ownership by teams rather than individuals.

According to SAFe, agile release trains should focus on value, not on projects [155]. Thus, knowledge about value for customers can be maintained in such release trains without additional documentation, even with normal amounts of staff turnover. SAFe also proposes the use of feature or component teams [157]. SAFe recommends supporting communities of practice, helping to share information in particular areas [155]. The creation of chapters or guilds has also been proposed [158] as a way to support and share specific topical knowledge. LeSS proposes something similar, encouraging experts to teach each other, and to create informal networks.

In addition, SAFe introduces the idea of enabler stories, user stories that are explicitly aimed for exploration. This allows teams to learn about a particular
5.5. CHALLENGES AND POTENTIAL SOLUTIONS (RQ2 AND RQ3)

LeSS promotes a similar practice by discussing reflection and encouraging improvement experiments. As a form of learning, LeSS also recommends specification by example, using concrete examples instead of more abstract user stories.

5.5.3.5 C3.e: Backward compatibility

C3.e: As a part of the maintained product knowledge, teams need to be aware of compatibility issues. In particular, as part of an agile way of working, it is challenging to maintain the knowledge of backwards compatibility as part of the requirements across different products and product versions of a product family.

Several of our companies have indicated the importance of backwards compatibility, particularly for customers.

"But also we use them (detailed requirements) for regression result, to make sure have we... it’s very important for our customers that we don’t change backward compatibility, we don’t change the behaviour without notifying our customers."

— T-APO

In some ways, using an agile way of working makes the preservation of backwards compatibility easier, as developers are given the freedom to handle changes in a way in which compatibility is not broken. However, in cases where the developer does not have this knowledge, if it is not somehow also captured via the requirements, compatibility may be broken.

"For me the part of being agile here is that we don’t define it (compatibility) on the highest level and just say you never break backward stability but you handle the changes. And of course you can only tolerate (this) to a certain degree. At some point it breaks legacy and then it goes to a level (where) we cannot release software anymore. You need to make sure that doesn’t happen."

— T-OPO

Potential Solution We have not found suggestions from the companies for this challenge as part of our interviews or focus groups. Although agility in general can help, the companies are lacking methods to capture backwards compatibility at a higher-level of abstraction, as captured by requirements. Furthermore, we find no potential solutions in SAFe/LeSS to address backward compatibility issues.

5.5.4 Representation of Requirements Knowledge

While the C3 challenges focus on internal understanding, the C4 challenges focus on how this knowledge can be effectively captured, managed and accessed.
5.5.4.1 C4.a: Manage levels vs. decomposition

In an agile environment, it is hard to map requirements to levels of decomposition. Classic levels (stakeholder, system, system element) do not fit with an agile way of working, since stakeholders can define low-level requirements. Yet the complexity of the software calls for some form of decomposition.

In Tech1WS, the participants explained that requirements from the customer express needs, and are very different from system requirements, which express elements of a solution. This company very much wants stakeholder requirements, they want to always define the problem before the solution, but customers sometimes provide them with detailed solutions instead of describing their problem. Although one can consider this a rather classical requirements problem, it is exacerbated by agility, which discourages many levels of requirements, and does not distinguish between different types of requirements.

In the ProcWS, we covered challenges in consistently breaking down requirements, particularly non-functional requirements, and expressed the need for more levels of classification. They made the point that breaking down requirements is very much experience-based, and is part of the process of building knowledge. Along the same lines, Manufacturing expressed the problem that requirements are often in the form of system requirements, focusing on a technical thing, and the real customer problem behind this requirement may be lost. They also echoed the challenge that those who describe problems (sales, customers) often have solutions in mind, meaning that the problem may not be captured. Telecom reported something similar, that although they have a means to capture the motivation behind requirements via a one-page slide, sometimes this step is skipped. Often their user stories (despite the name) tend to be more technically focused, and the user value is only implicit.

Potential Solution One potential solution offered by Technology 1 is to allow stakeholders to specify requirements on any level of abstraction, but then use matching between levels to match details to motivations, or point out missing requirements at one level. They also suggest to support distributed requirements analysis, so that stakeholder with different expertise (e.g., problem, solution) can contribute. Traceability was offered as a solution by both Processing and Technology 2, allowing for a requirements structure linked via traceability, bearing similarity to the suggestion above.

Automotive 2 emphasized the importance of the interplay between requirements and architecture. They suggested to distinguish dimensioning functional requirements (e.g., in form of use cases) and quality attributes and to use both separately as input for a suitable architectural decomposition. As is common practice with quality attributes and quality scenarios, such dimensioning functional requirements would be carefully chosen early on as typical representatives and could be used to reason whether a given architecture is suitable to support such functionality, thus functioning as a blue print for architectural decomposition.

SAFe suggests having a clear hierarchy of people and roles, from enterprise level to team \[155\] p.70, complemented by a hierarchical view of requirements in
5.5. CHALLENGES AND POTENTIAL SOLUTIONS (RQ2 AND RQ3)

four levels, from epics to capabilities to features to user stories that corresponds to this clear hierarchy [155, p.177-6,104-109]. Based on this hierarchy and requirements information model, SAFe advises to transport the stakeholder view to components across potential hierarchies [157]. LeSS also provides various splitting and refinement guides using requirement areas, major areas of customer concern, where each area has its own backlog and feature teams [154, p.30]. LeSS advises for traceability of certain items (e.g., product backlog items have requirement area attributes which trace to their associated area) [154, p.216]. However, they advise splitting requirements only to three levels [154, p.222].

Overall, we see potential solutions from both industry and the literature; however, although recommendations are given for limited refinement and traceability, this issue of customer vs. system requirements is not deeply addressed. It seems SAFe and LeSS may advise to avoid purely system requirements with no links to customer rationale, which does not appear to be good advice for our case companies.

5.5.4.2  C4.b: Quality requirements as thresholds

Often quality targets are within a range. Negotiation of cost-value trade-off is difficult to capture and manage with current representations.

As agile methods recommend various levels of user stories, Tech1WS reported issues with using such presentations for quantitative quality requirements trade-offs. For them, quality requirements are thresholds, and it often takes a lot of time to quantify thresholds for requirements, leaving ‘TBD’ in the meantime. The systems and representations the company has now are not capable of dealing with these type of thresholds, and they default to a single hard target for requirements. There is also a need for guidance in how to find these boundaries, a process as part of requirements specification. Other companies confirm this challenge.

Potential Solution  The companies offered a few solutions to this challenge. For example, Telecom discusses trade-offs when they are brought up by a team, and these trade-offs are peer-reviewed among teams and system managers in gerrit (a code collaboration tool). Technology 1 emphasised that thresholds for quality requirements can be a good way to indicate and moderate price negotiations between different development partners.

In SAFe, non-functional requirements [4] are constraints on a program level, constraining the backlogs at every level (system, feature, team) [157, p.77,79]. We find no potential solution in LeSS to help with quality requirements.

Overall, one can argue that this issue may occur also in a non-agile context, but use of user stories makes solving this issue more challenging, and current scaled agile frameworks do not offer any specific solutions for it.

---

[4] For our purposes we treat NFRs and quality requirements as the same, a more detailed debate on this is out of our scope.
5.5.4.3 C4.c: Tooling not fit for purpose

Tooling plays a significant role in agile processes, but available tools are often not designed to support large-scale agile practices. Part of the problem is access to requirements, as traditionally tools do not allow access to all requirements, but some form of managed access is often needed.

Agile endeavours to empower teams, but it is challenging to determine the scope of this power. Technology 1 has expressed a requirements/tooling access challenge where teams rely on requirements they do not have view or edit access to. These requirements are exported outside of the tool to other formats for them to use as inputs to their process. This requires extra effort and results in inconsistent requirements, but the alternative, to let every team access and edit/refine all requirements, would need to be carefully managed both in terms of processes and tools. Often, teams lack expertise and knowledge to modify requirements that they have not worked closely with, even if they are dependent on those requirements, and would like changes.

This brings up a broader challenge related to the need for specialized tools. Automotive 1 have described their used of tooling, at the moment they use traditional tools for requirements management (e.g., Doors), and tools that are aimed for agile (e.g., JIRA). However, these tools are largely separated and not designed to fit a large-scale agile process.

Similarly, in Telecom, current tooling was brought up as a hindrance for speed and agility. Interviewees described the current process of updating system requirements as too slow and cumbersome. They stated that by introducing a more efficient tool solution, engineers could potentially be more motivated to make changes to requirements and by this narrow the gap between agile user stories and requirements.

The need of a tool-chain that better supports agile information flows was further confirmed by other companies.

Potential Solution There are a few solutions to this challenge suggested by industry: Technology 2 suggests that only the product owner updates requirements, and all requests must go through them. Tooling should be updated to support this model. This imposes governance, but may create bottlenecks if change requests are frequent. In Automotive 1, 2, and Telecom, it is suggested that the team itself takes the main responsibility to update the requirements. This approach removes bottlenecks, but makes governance more difficult and requires strong support from tooling, especially when knowledge of updates needs to be shared across teams and abstraction levels.

SAFe offers no solutions to this issue. LeSS advises against using software tools for sprint backlogs [154, p. 18, 281], but discusses tooling for large product backlogs such as boards, wikis, pictures and spreadsheets [154, p. 23, 210]. However, the source does not discuss the issue of tool or requirements access.

To summarize, our companies of study are looking for better tooling and more effective requirements access solutions, and while some custom tools show promise, SAFe and LeSS do not emphasize tooling or discuss access control.
### 5.5.4.4 C4.d: Accommodate different representations

Individual teams strive to tailor requirements related artifacts to what works best in their context. This however is seemingly in conflict with the system level goal of keeping artifacts consistent and manageable. Companies experience a lack of support for navigating this conflict.

In Tech1WS, the participants expressed frustration with this challenge. On the one hand, specific teams use a variety of different representations for requirements depending on purpose. Word documents are used for quick exchange with external stakeholders, figures, graphs, and models are used to discuss, and teams use them in the way most promising to get the job done. On the other hand, a consistent view on system level needs to be arranged and there are reasons to limit the flexibility in representations. This is partly due to technical reasons (it is easier to store text requirements in a requirements database, see also Challenge 4.c tooling not fit for purpose in Section 5.5.4.3) and partly due to organizational reasons (system-level planning demands that certain information is easily accessible, it is easier to share a small number of simple formats across a large organization).

This trade-off demands for an approach that allows starting from a consistent requirements model of the full system, quickly draft sketches in arbitrary representations and coordinate between teams and external stakeholders, and then re-integrate any knowledge gained in the consistent requirements model to evaluate it in the context of system or platform variability constraints.

**Potential solution** We did not find potential solutions to this issue from our participating companies as part of current workshops or interviews. SAFe advocates for teams to have their own individual user story flavor [155], which to some degree supports freedom in using a format that best fits local work. This goes along with SAFe’s tendency to emphasize team independence, where teams are responsible for their own way of working. However, SAFe does recommend some sharing of knowledge and practices using book clubs and guilds [158]. We find no potential solutions for different representations of requirements in LeSS.

### 5.5.4.5 C4.e: Consistent requirements quality

The quality of requirements artifacts (i.e. user stories, backlogs) differs (e.g. level of detail). This makes working with requirements at higher levels, across teams or boundaries, difficult.

Telecom reports that the quality of requirements differs from system to system, or between roles and sites. They find that the quality of user stories also varies, sometimes they are from the perspective of the user, while often they are phrased like a technical task. Technology 1 reports similar findings, with backlogs from different teams relating to the same product or platform having very different styles. Processing similarly reports that they lack a common way of working with requirements, which means that some teams try to minimize the
requirements they write, while other teams try to specify everything, defining similar requirements over and over across projects or backlogs.

**Potential solution** Manufacturing expresses the importance of experience in operationalizing requirements in an effective way. As such, either skilled personnel or training may be required. Currently, Telecom are exploring supporting requirements reviews using T-Reqs and gerrit.

SAFe advocates responsibility for Ways of Working, and other practices such as book clubs and guilds [158], potentially sharing knowledge on ways of working with requirements or ideas on requirements quality. Similarly, SAFe supports the formation of community of practices to align on needs [155, p.25,43,290]. Less offers no potential solution to help with requirements quality. Overall, some practices are suggested by our industry sources and the large-scale agile literature, but the challenge is not yet sufficiently addressed.

## 5.5.5 Process Aspects

### 5.5.5.1 C5.a: Prioritization of distributed functionality

The frequency of dependencies in large-scale agile makes prioritization of products or requirements between teams difficult. Bottom-up prioritization is not working well, since teams tend to start with simpler tasks.

In TelWS, participants reported that, before their agile transition, they had spent a lot of time analyzing features that did not end up in the product. They see an improvement on this using a more agile approach. However, a potential drawback brought up at our case companies relates to features that have a scale at which many teams or even several release trains need to be included. Each of these teams or release trains usually has a full backlog and when coordinating functionality, each involved party has their own critical parts to consider.

In ManWS, participants complained that developers often do not take on the highest priority task first, often because they are lacking expertise. When prioritizing their backlog, teams consider which of the tasks they can do in the time available provides most value. A particular complex task may therefore not be touched, since the team considers the time to implement it to be in no good relation to the value they could provide with other tasks. This can be a problem when considering highly complex products. A complex feature then might take a very long time to be deployed, since teams go for “lower hanging fruits” and are unable to consider the cost this delay creates. This becomes a problem, especially if other teams or release trains have already committed to develop their part of the complex feature, as their effort then does not generate business value (since the overall feature is still incomplete).

Simplifying, one could summarize this as: Letting individual teams prioritize (bottom-up prioritization) is not working well, as teams tend to favor simple tasks.
Potential solution Although our participating companies have prioritization with coordination as a challenge, they also offer several potential solutions. Manufacturing prioritizes their release backlog based on a business dashboard, including items such as commitments to customer. Automotive 2 aims to address this challenge by introducing a clear product owner hierarchy and puts the focus on interfaces instead of requirements. This helps address prioritization as teams can articulate their needs towards other teams and interface issues can be addressed with high priority. Through appropriate architectural decomposition, complex requirements will inform changes on interface definitions and concrete requirements on team level. Processing has developed a system with some success, prioritizing by risk, and working on the next part with the highest technical risk. They calculate risk via a system design meeting focusing on technical needs. They also have forums with both business and technical experts, focusing on a bidirectional flow between both roles. Technical risks are transferred in technical review meetings, helping awareness.

The large-scale agile frameworks also offer some solutions. SAFe describes techniques such as combining “weighted shortest job first”, “portfolio backlog”, and “program kanban” to support cross-cutting initiatives towards prioritization \[155\] p.65,104,212]. It also advocates the combination of team backlog, business values and an “interaction backlog” \[155\] p.109,127,137] and sequencing tasks based on the cost of delay [155] p.175]. However, it’s not clear how to combine all these suggestions together into one coordinated process. LeSS recommends that one product owner acts as a single source of prioritization, and that a multi-site product backlog review is used to help prioritization across products \[154\]. Overall, this is a challenge that comes with a lot of potential solutions; however, there is a lack of empirical evidence or proven strategies to inform companies on which approach may work best in a particular context.

5.5.5.2 C5.b: Manage completeness

In a large-scale agile context, it is not clear when requirements are complete enough. It is also not clear on what level to judge completeness: per sprint? per product? per system? Which view is the most important for completeness?

The ProcWS participants described their agile transition in relation to requirements completeness goals. Initially, they wanted to have complete requirements, but it was difficult to have an overview. They questioned how many requirements they could manage in a sprint. The ManWS participants expressed similar difficulties with their specification of system control, the requirements could not cover everything, but instead focused on the algorithm and control.

Potential solution Generally, for our companies that mentioned this as an issue, the goal of complete requirements was relaxed for something more manageable. SAFe offers no potential solution, in fact, by advocating building incrementally and producing minimal viable products (MVPs) as principles,
SAFe actually recommends against requirements completeness \[155\,\text{p.}\,77\]. Similarly, LeSS recommends to “take a bite”, analyzing and implementing small parts of the problem, forgoing completeness in requirements \[154\,\text{p.}\,3\,202\].

Taking an agile mindset, one can argue that requirements completeness should not be a goal; however, at least two of our companies have struggled with this even in an agile context. Even working incrementally, it’s not clear how complete or detailed requirements for an increment should be.

### 5.5.5.3 C5.c: Consistent requirements processes

Different teams create and manage their requirements using different processes, tools and level of detail. Coordination and sharing is difficult.

The TelWS participants indicated that distributed development in different countries with different cultures has made consistency in requirements processes difficult. Some locations are embracing agile, with others still want a more procedural approach with document approval. Processing has addressed similar frustrations, but with tooling. People are reluctant to stop using common tools like Excel and migrate to modern requirements management tools. The result ranges from full to partial migration, sometimes moving between the tool and Excel, with the Excel version updated more frequently. The problem persists due to usability complaints about the new tool as well as management not enforcing its use. If some migrate and some do not, inconsistency is the result, hampering coordination. Technology 1 sees similar tool-related migration and consistency problems. This challenge relates strongly to both Challenge 4.c (requirements quality) and 4.e (tooling).

**Potential solution** Telecom suggested that requirements consistency could be managed similarly to code and test consistency. Therefore, they aimed for a system where requirements are stored in the same repository as code and test, be consistently updated during sprints, and peer-reviewed to ensure comparability. This would then lead to a more consistent requirements process.

With respect to SAFe, the recommendation for a clear hierarchy, from the enterprise level to team, may help to promote consistent processes \[155\,157\]. However, SAFe also advocates for team independence, with responsibility for their own way of working. Coordination mechanisms like book clubs and guilds can help to share best practices, even given independence between teams \[158\]. LeSS does not appear to directly address this issue.

Overall, the level of consistency needed between teams is an open question, likely depending on context.

### 5.5.5.4 C5.d: Quality vs time-to-market:

It is often not clear what quality level (of requirements, products, deliverables) is good enough. It is not clear when to continue improving or when to release, particularly on a large-scale.
Given the aim to shorten time-to-market, answering the question of what is good-enough quality is becoming a challenge for our case companies. This challenge holds for the releases of the actual product, but also for intermediate deliverables, such as requirements. It is one of the agile dogmas that one should not invest time into high-quality specifications of requirements that then might never be implemented. The same holds for products: there is a widely spread idea that it is better to have a first version of the product in the market and then iteratively improve it to achieve good fitness for purpose without overshooting the required quality.

This view however raises practical concerns, especially when embracing agility at scale. Telecom and Automotive 2 did indicate that sometimes requirements were not of sufficient quality to allow testing. While in small-scale agile, this could be mitigated through intra-team communication, at large-scale there must be a form of moderation to ensure that lack of information can be articulated and fixed between teams and release trains.

Technology 1 indicated a related challenge with respect to releases of products. Obviously, quality comes at a price which could manifest in product cost, time-to-delivery, or a combination of both. However, it is very hard to discuss this with customers, since for large-scale products it is difficult to make customers aware of the price. Thus, when discussing with customers about quantitative quality requirements on systems that include hardware or software components, customers usually strongly demand very high quality, even beyond what they actually need or can afford for a concrete business case.

Often, there is also a reluctance to record a number on quality requirements, as then there is a level of commitment for this number, when agility demands flexibility. Yet, for many qualities, it will be difficult to address them late in the development without careful planning. While for example functionality can be added later, fixing major performance or usability problems late can entail major refactoring and rework. In addition, customer expectations about quality remain rather constant.

**Potential solution** Relying on frequent reviews of requirements as part of sprint deliverables can be a good way to establish a feedback channel about requirements quality and lack of information, as brought up by Telecom. This could help teams to find over time a good balance on providing just enough requirement quality: missing information might delay development, while to elaborate requirements will unnecessarily lengthen time-to-market.

Safe advocates for built-in-quality as part of the agile process [155, p.23, 140], and provides guidance for reducing time-to-market through value stream mapping [155, p.298], but does not explicitly address the trade offs between time and quality. We did not identify concrete guidance for this challenge within LeSS.

Generally, there is a trade-off between product quality and time-to-market, for products and releases as well as for deliverables and artifacts needed during development. There is a lack of guidance to balance this trade-off.
5.5.6 Organizational Aspects

5.5.6.1 C6.a: Bridge Plan-Driven and Agile

It is hard to bridge the gap between plan-driven, document-centric approaches on system level and value-driven, agile approaches on team level. Companies struggle to stay proactive on system level as well as to leverage knowledge about requirements that is generated on team level.

From a product perspective, a plan-driven or stage-gate approach is important. Release of a new product needs to be planned and longer development cycles for hardware and mechanical components need to be scheduled. All of our case companies have agile software development teams that operate within the context of a larger system engineering process, which one interviewee described as agile islands:

“It feels like agile islands in a waterfall.” — FG 2

The challenge we found here regardless of agile scope in the specific case is continuous information exchange between plan-driven and agile parts of an organization. Incubation of new innovative ideas, facilitating quick feedback loops, and quick learning on potential business value are important assets to remain competitive, yet they are hard to integrate into the overall system development approach in all our cases.

In the Telecom case, we found that system managers feel disconnected from the agile teams. Their role is to be experts on a certain part of the system and support teams with their knowledge of the system requirements. However, as one interviewee stated they currently cannot be in contact with all teams and might therefore not get a notification if something has been changed with respect to existing requirements.

“If [...] a team updates a past requirement, perhaps I should get like a notification on that so I can ask them ‘Have you forgotten X?’.” — T-SysM

Similar challenges exist with the other companies, e.g. in the Automotive 2 case where agile teams can add new backlog items or change existing ones in collaboration with the product owner. However, since agile teams do not interact directly with system requirements (see b) creating and maintaining traces), they do not consider knowledge about them to be of importance. Further, backlog items are easy to understand, even for stakeholders not directly involved, and allow them to share their opinion. While this is generally perceived positively by the interviewees, it was also brought up that this can cause the function owner to be overexposed to change requests. One function owner expressed this as follows.

“The more people look into requirements, the more they read them, the more iterations it will become. [...] there is going to be more opinions, comments and also more work.” — A2-FO

As this can lead to inconsistencies between changed and new backlog items and the system requirements, e.g., in the case where a system requirement related to a new user story already existed, increased gate-keeping becomes necessary.
This generates effort for backlog grooming by the (agile) product owner, and managing of system requirements by the (plan-driven) function owner. The current separation between both worlds does not seem to be ideal, since product and function owner can easily become bottlenecks, and late resolution of inconsistencies can create additional effort. If the actual implementation deviates from the original requirement or when some requirements are not implemented, this will surface as problems during system integration and testing. Tests are developed against the plan-driven requirements and are therefore in need of an up-to-date version.

“If I have a requirement saying this thing should happen, when I test it, I find out that what is supposed to happen doesn’t happen. [...] And then I find out the requirement wasn’t updated. So actually the implementation was correct but the requirement isn’t matching the implementation.” — A2-ST

Further, if the system has to be evolved or maintained in the future, outdated requirements can cause misunderstandings.

Potential solution In our focus groups, the governance of requirements between system level planning and agile teams was raised as a key issue. Telecom emphasized that the team should be enabled to update the requirements during sprints, similarly to source code, tests, and documentation. T-Req as a tool solution was again mentioned as potential enabler[150]. In contrast, Technology 2 placed the responsibility of updating requirements with the product owner. While we did not identify related practices in LeSS, we believe that SAFe offers good advice on governance of requirements and related knowledge across levels in that it provides a clear hierarchy from enterprise level to individual teams[155,157]. Yet, mastering this part will require significant effort by any company transitioning into agile development, as we found few concrete practices and guidelines in the agile frameworks.

While transitioning from plan-driven to large-scale agile, companies start to rethink the role of systems engineering artifacts. While many of these artifacts (including requirements specifications on various levels) have been static documents, agile development now demands for actively managed artifacts that help with the coordination of agile teams within a plan-driven system engineering organization. We believe that this will be a challenge even for fully agile system development.

5.5.6.2 C6.b: Plan V & V based on requirements

In the past, verification and validation (V & V) was planned based on requirements. Now that requirements are inherently incomplete and incremental throughout development, how does one plan for testing? Particularly, it is hard to provide guidelines and traceability, to allocate resources, manage test artifact information for decision making, and align requirements with system tests.

Using the V model, our case companies were used to a tight link between requirements and tests. As the nature of requirements changes, these links must be rethought. However, planning of test activities is still critical, particularly
in order to allocate resources (time, hardware, people, etc.), and can be costly. Manufacturing has indicated that following new agile practices means there is only partial traceability between tests and requirements. High-level artifacts are used to define test guidelines (boundaries on tests); however, these are hard to follow, as for example, a wide range of test oracles needs to be taken into account, which vary from numbers (dimensions or signals) to user action responses. More guidelines are needed in testing. In this company, system engineers are responsible for the V & V strategy, while the project managers, who are more in line with the agile processes, do not do such planning. This gap between agility and the testing team causes challenges.

More generally, this requires to rethink traceability and one has to discuss the different information items in relation to the roles that use them. In our large-scale agile system development cases, we find a very complex picture and it is partially unclear how information items relate and which stakeholder needs could be satisfied through traceability. Several interviewees in the Telecom case stated that their system requirements work as a documentation of what the system is doing, rather than a plan of what shall be implemented.

“You can’t really afford to have this kind of static requirements work upfront which will be a waste anyway when you implement stuff. The way we handle requirements now is more like a system description.” — T-TA

Yet, as mentioned before, user stories and tests are not enough (C3.a), thus there is a need to document any assumptions or decisions taken during testing, which can be interpreted as requirements that the system should fulfil from now on.

Potential solution Manufacturing has partially addressed this challenge by establishing virtual test rigs and simulation models, reducing the cost of testing and at the same time making testing infrastructure more directly available to development teams. Telecom suggested to include system requirements in the same repository as code and tests and to make sure that the same quality assurance mechanisms are applied. Ideally, this ensures that tests and requirements are consistent, as they are modified at the same time, and traced, as they share the same commit as well as explicit trace links.

SAFe recommends duality in backlog items and tests, and describes solution intents as linking specifications to tests [155, p.187]. Adopting a more cross-functional organization, including testers or system engineers in the agile teams, would also help to alleviate these issues [155, p.97]. Less does not offer any specific solution here.

Despite these promising suggestions, the planning of validation and verification remains a huge challenge especially in system engineering and its various disciplines. Empirical evidence about the proposed practices and proven approaches are currently lacking.

5.5.6.3 C6.c: Time for invention and planning

Research activities and exploration are hard to fit into development sprints but offer fundamental information towards requirements. Which roles should be involved at which time?
Study participants in Automotive 1 reported that an exploration of solution space is difficult within agile sprints, as it would be impossible to commit to a fixed schedule without deep knowledge about new features. Pre-development is required to better understand the impact of new features. If this is done by a dedicated group, this would imply documentation and hand-over of results and slow down the process. If it were done by the developers the development process would be slowed.

**Potential solution** As a remedy, specific exploration sprints were brought up. Another solution could be to transfer engineers between pre-development and agile system development, so that they can also share their knowledge with team members.

SAFe again recommends capturing the solution intent, including a repository of current and future solution behaviours [155 p.20]. SAFe also describes both enabler stories, stories that explicitly support exploration [155 p.108] and iterations dedicated to innovation and planning [155 p.96,154]. LeSS offers no potential solution.

In summary, there is a lack of experience or evidence with respect to the proposed practices. For a company transitioning towards large-scale agile, this challenge requires careful scoping of agility within system development.

### 5.5.6.4 C6.d: Impact on infrastructure

When neglecting upfront analysis, the impact on infrastructure might become obvious too late. Then, updating infrastructure (e.g., improving labs for testing) increases cycle time and time-to-market.

In system development, integration testing often depends on a strong laboratory setup that allows testing hardware, software, and potentially mechanics together. Although this relates to challenge C6.b, required infrastructure changes may go beyond testing infrastructure. While a new feature might mainly depend on changes of software and can be provided in an incremental, fast-paced way, it could require an update of the test environment, which may include sophisticated hardware and environment models. However, changing the test environment might take as long as finishing the software components, thus introducing delays, if not started in due time. Similar concerns relate to other infrastructure for continuous integration, delivery, and deployment.

**Potential solution** From a testing prospective, as mentioned in C6.b, companies can make use of virtual test rigs and simulation models to avoid physical infrastructure changes. Peer-reviewing of requirements can raise awareness about potential impact on infrastructure early on.

SAFe recommends having a cross-functional organization, which can help teams to understand the wider impact of their features and changes, including impact on infrastructure [155 p.97]. We do not find a potential solution in LeSS.

This challenge shows that independent of the pervasiveness in Fig. 5.3, there is a need to maintain a system-level perspective beyond self-organized
Table 5.3: Summary of results for Challenge Area 1: Build and maintain shared understanding of customer value.

<table>
<thead>
<tr>
<th>ID</th>
<th>Challenge</th>
<th>Proposed practices from case companies</th>
<th>Proposed practices from SAFe</th>
<th>Proposed practices from LeSS</th>
<th>Research gap in large-scale agile</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.a</td>
<td>Bridge gap to customer</td>
<td>Visualization of requirements to facilitate discussion Reduce handovers, XFTs across levels Keep product management close Focus on value every sprint</td>
<td>Frequent (train) demos Customer involved at every level Teams take economic view Team covers all necessary roles</td>
<td>Customer-centric Sprint-Review bazaar PO connects teams / customer</td>
<td>Provide concrete advice and tools for establishing, managing, and validating shared understanding of customer value.</td>
</tr>
<tr>
<td>1.b</td>
<td>Building long-lasting customer knowledge</td>
<td>-</td>
<td>Feature teams Component teams</td>
<td>-</td>
<td>Research gap similar to 1.a, but with long-term memory in mind.</td>
</tr>
</tbody>
</table>

5.6 Discussion and Implications

Even though the seven cases differ in their context, i.e., domain and pervasiveness of agile methods within system development, we found common concerns and challenges with respect to RE. As our investigation reveals, systems companies face severe challenges that are not sufficiently covered by common large-scale agile frameworks. Generally this suggests that in order to yield their full benefits, agile practices must be combined with a sufficiently strong mechanism to manage requirements and related knowledge. We found challenges in six different areas and while we could derive potential solutions from data collected with our case companies as well as from our analysis of agile frameworks, we see a significant need for future research. We will discuss each of the challenge areas and their implications for future research in the following.

5.6.1 Build and Maintain Shared Understanding of Customer Value

Managing customer value is usually assumed to be the core strengths of agile approaches and we identified potential solutions both in LeSS and SAFe. Yet, we found in all our case companies that the distance between the customers
and the development is perceived to be too large (summarized in Table 5.3). In particular, it was described as difficult to break down a feature request into small packages that both have customer value and can be delivered in small iterations. However, agile values such as individuals and interactions [93] as well as agile practices such as continuous delivery [94] depend on a good notion of value. Yet, we found this particularly hard to establish in large-scale system development, because of unclear customer role and scale. The customer role is often unclear, since development teams do not only need to produce value to external customers, but also to other roles within the company, e.g., in order to prepare for maintenance.

In case of an external customer, any customer-visible feature will imply more work than can be done within one sprint or by one team, at the scale of our case companies. This makes feature decomposition necessary and it is impossible for a single team to demonstrate customer value at the end of a typical sprint. Related work in this direction has, in particular, pointed out challenges with the practice of customer representatives [15, 16, 19], but it seems that the notion of value itself is problematic and a shared language for discussing value is needed [94, 159, 160] as well as approaches to systematically enable, build, and assess shared understanding [161]. Without those concepts, our case companies struggle to establish, manage, and validate a shared understanding of customer value throughout the development organization and we see the need for future research to address these challenges.

### 5.6.2 Support Change and Evolution

As summarized in Table 5.4, our results indicate that sufficient facilities for updating system requirements based on agile learning are currently missing affecting managing experimental requirements, synchronizing development, re-using requirements, and managing the lifecycle of requirements. Thus, such updates are a result of manual work, leading to inconsistencies, which are expensive to remove and can be considered waste in the overall development process. In addition, developers have little intrinsic motivation to update requirements models based on updates to user stories, as they are not part of their delivery (usually code and tests). If, however, requirements updates were not propagated, the system requirements view would become quickly obsolete and detached from the real system. Consequently, roles responsible for customer and high-level system requirements (product owners, function owners, system managers) fear a loss of important knowledge for later maintenance of the systems. A more systematic approach to manage requirements updates received from agile teams would make their jobs much easier.

We believe that more research on these aspects is urgently needed to provide better guidance, approaches and tools to manage evolving requirements. In line with Cockburn, we believe that agility is a game on two levels: not only should one aim to deploy features to the market quickly, but one should also increase the organizations ability to provide value to customers in the future [162].
Table 5.4: Summary of results for Challenge Area 2: Support change and evolution.

<table>
<thead>
<tr>
<th>ID</th>
<th>Challenge</th>
<th>Proposed practices from case companies</th>
<th>Proposed practices from SAFe</th>
<th>Proposed practices from LeSS</th>
<th>Research gap in large-scale agile</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.a</td>
<td>Managing experimental requirements</td>
<td>Introduce Learning and increment planning Sprints; T-Reqs: Use git branching and merging;</td>
<td>Combine enabler stories, architecture, and exploration; Set-based design; (Variable) solution intent</td>
<td>Use backlog</td>
<td>Design and evaluate an approach to manage experimental requirements.</td>
</tr>
<tr>
<td>2.b</td>
<td>Synchronization of development</td>
<td>Differing levels of agile pervasiveness</td>
<td>Biweekly ART sync, f2f PI planning; Enabler stories (Exploration); Tribes, chapters, guilds; Unity hour</td>
<td>Continuous improvement; Retrospectives (Team+overall)</td>
<td>Design and evaluate an approach to synchronize development based on promising ideas in literature.</td>
</tr>
<tr>
<td>2.c</td>
<td>Avoid re-specifying, encourage re-use</td>
<td>Product-line engineering; Move from project to product focus</td>
<td>Product-focused, Set based design</td>
<td>Product-focused, Avoid duplicate product functionality, avoid narrow product definition</td>
<td>Strategies and guidance for systematic reuse and agile product-line engineering at scale</td>
</tr>
<tr>
<td>2.d</td>
<td>Updating requirements</td>
<td>T-Reqs: Reviews supported by git and gerrit</td>
<td>Guilds and chapters</td>
<td>Requirement areas will change, will get less important, have a lifecycle, be retired</td>
<td>Provide concrete advice and tools for updating requirements and to establish awareness of the current state.</td>
</tr>
</tbody>
</table>

5.6.3 Build and Maintain Shared Understanding about System

Our third challenge area relates to building and maintaining a shared understanding about the system and is summarized in Table 5.5. Historically, plan-driven approaches suggest to distinguish between requirements specified from a user perspective (user or customer requirements specification) and those specified from a system perspective (system or supplier requirements specification) [163]. Agile methods mainly concern themselves with customer or user value, thus covering the content of a user requirements specification and even going beyond by focusing on the value that is generated for users and customers. There is virtue in such value- or problem-based specifications [164][165], and we
Table 5.5: Summary of results for Challenge Area 3: Build and maintain shared understanding about system.

<table>
<thead>
<tr>
<th>ID</th>
<th>Challenge</th>
<th>Proposed practices from case companies</th>
<th>Proposed practices from SAFe</th>
<th>Proposed practices from LeSS</th>
<th>Research gap in large-scale agile</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.a</td>
<td>Documentation to complement tests and stories</td>
<td>Use models for interfaces and behaviors, additional text summaries</td>
<td>Use models to analyse requirements</td>
<td>-</td>
<td>Methods to capture comprehensible big picture of agile requirements and motivations</td>
</tr>
<tr>
<td>3.b</td>
<td>System vs. component thinking;</td>
<td>Need (global) baseline; Requirements for product, not organization; Establish ownership on all levels; Architects on each team</td>
<td>Clear breakdown from enterprise level to team Feature and component teams Systems thinking, Tribal unity, communicating the vision</td>
<td>Principle: Whole product focus Multi-team product backlog refinement PO engages team to own product</td>
<td>Provide and evaluate concrete advice and tools to support systems thinking on all levels as well as governance of requirements.</td>
</tr>
<tr>
<td>3.c</td>
<td>Creating and maintaining traces</td>
<td>Reuse features( = groups of reqs); Reusable modules (= requirements and solutions); Move from project to product focus</td>
<td>Describe the solution (Documentation)</td>
<td>Link to wiki pages Backlog items to ancestors, max. 3 levels</td>
<td>Provide guidance and tools for large-scale agile traceability</td>
</tr>
<tr>
<td>3.d</td>
<td>Learning and long-term knowledge</td>
<td>Exploit ownership of feature and tools to document less</td>
<td>ART focus on value, not project Enabler stories (Exploration) Feature and Component teams Community of practice, chapters/guilds</td>
<td>Reflection + improvement experiments Experts teach each other, informal networks Specification by example</td>
<td>New approaches towards requirements as a knowledge management problem</td>
</tr>
<tr>
<td>3.e</td>
<td>Backward compatibility</td>
<td>Push responsibility (and freedom) to developer</td>
<td>-</td>
<td>-</td>
<td>Strategies and guidance for systematic management of backwards compatibility.</td>
</tr>
</tbody>
</table>

agree that user or customer value is an important knowledge area with respect to requirements. We found, however, that system requirements knowledge is crucial for large-scale system development as well, especially considering the
very long maintenance cycles.

We generally find this perspective of requirements with a particular system or solution in mind to be underrepresented in scaled agile frameworks. User stories have been found insufficient to cover such knowledge and (automated) test cases are often named as an alternative, especially for small-scale projects. Our findings suggest that using test cases, even in combination with user stories, is not sufficient, in particular with respect to supporting the understanding of a system’s current functionality. Specifically, we identify a lack of guidance in agile frameworks with respect to capturing a comprehensive big picture of requirements and their rationale, a finding in agreement with Heikkila et al. Because of this lack it is challenging to support systems thinking and requirements governance, to provide (often required) traceability, and to manage long-term knowledge as well as backwards compatibility. Therefore, we see the need for more work investigating the use of different notations, techniques or methods to inform early analysis of incoming requirements.

Even though such documentation and management of system requirement may feel non-agile by nature, it becomes crucial to support agile systems development. While significant work exists in the area of agile modeling, our focus companies do not report experiences with these solutions. We distinguish therefore between agile requirements engineering as covered in most of the related work and requirements engineering for agile system-development (RE4agile), where we do not require an agile approach to engineering requirements. We see RE4agile as a fundamental service to provide crucial requirements knowledge so that agile teams can perform. Our findings suggest that such support cannot be offered sufficiently by traditional, upfront RE, as indicated. Similarly, we did not find any specific roles that emerge in the large-scale agile environment comparable to the roles presented in.

Our results suggest that continuous and agile development methods on a large scale require new concepts. Hybrid approaches that aim to combine strengths of both plan-driven (waterfall) and value-driven (agile) paradigms may offer inspiration, but are at this point not sufficiently documented through empirical studies to relate them to our findings. In more recent parallel work with the same companies, we have been exploring theories and methods to manage and govern shared objects such as requirements, architecture descriptions, APIs, and user documentation. Interpreting such items as boundary objects can help link individual teams to shared views of vision of the whole system and ultimately lead to effective agile approaches to manage such knowledge at scale. We have explored this approach with respect to strategic API management and governance and we are confident that this research direction will yield useful concepts and theories to tackle this challenge area.

5.6.4 Representation of requirements knowledge

The fourth challenge area includes challenges that relate to the representation of requirements knowledge (see Table 5.6). One underlying observation of the challenges in this area relates to the shared responsibility for requirements knowledge. In particular, scaled agile appears to imply that teams take more responsibility for both customer and system requirements. This in turn implies
a bi-directional flow of requirements knowledge. On the one hand, it must both be relayed top-down, from system level planning to teams. On the other hand, it must flow bottom-up, from teams that explore the best way of satisfying a customer need through incremental and iterative work. We discovered challenges with both directions.

Bottom-up, the current tooling is not fit for purpose, since it does not allow teams to create and share knowledge efficiently. In addition, teams are expected to take responsibility for their own ways of working and to establish suitable flavours of requirements artifacts. How can individual teams have their own specialized requirements representations and still relate to the overall system-level requirements model? Top-down, a suitable decomposition of requirements is hard to achieve, especially since agile frameworks do not cover the duality of customer and system requirements. It is also difficult to establish a consistent requirement quality. Related work by Wohlrab et al. suggests that diversity and alignment of representations can be balanced, especially when taking into account information and consistency needs on different levels of abstractions and at different times during the development cycle \cite{170}.

Challenges with this shared responsibility for generating and managing requirements-related knowledge surfaces in difficulties to establish thresholds for quality requirements. How does a large-scale organization align on such threshold and manage their evolution when new knowledge becomes available?

We have been evaluating the use of T-Reqs, an approach to manage textual requirements in git version control together with tests and source code \cite{150}. Custom tools such as T-Reqs can be accessible across an organization, and allow for customized access to requirements and peer-reviews of requirements changes by other teams and system managers. Existing work in the requirements literature has recognized that user story quality in practice can be problematic and has introduced various quality frameworks and tools to manually and automatically detect quality issues, e.g., \cite{171}, which in our view would integrate well with such peer-reviews. Such an approach promises to help with giving teams access to requirements tooling, supporting quality assurance, and even with re-negotiating quality thresholds and we encourage further research in this area.

As our results suggest, it is crucial to establish suitable exchange and management of knowledge throughout large-scale agile system development. Agile development works best with a continuous inflow of new requirements and can in turn help to resolve ambiguities and refine requirements just in time, as new knowledge becomes available. However, it is important to support updating system requirements models and to coordinate the information flow between parallel teams.

This finding suggests that communication issues continue to be relevant in large-scale agile RE, in contrast to what is suggested by related studies, e.g., \cite{15,54}.

### 5.6.5 Process aspects

Our fifth challenge area relates to the process of working with requirements. As the previous challenge areas indicate that requirements knowledge is not only continuously evolving, but also spread between customer value and system
requirements as well as between a consistent requirements model of the complete system and specialized views of individual teams, it becomes clear that strong, continuous, and distributed processes must be established. Within this problem-space, the well-known challenge of just enough requirements engineering reappears with force: how can a developing organization with dozens of agile teams find this fine balance where time-to-market is neither impacted by too much missing information nor excessive requirements work?

A concrete challenge relates to distributed prioritization. While this is certainly challenging, it appears that prioritization by risk rather than value can be a good practice in many cases. This suggestion is in line with recent research by Hadar et al., suggesting to use risk for prioritization. They suggest that risk is in many cases easier to quantify than value, thus providing a strong prioritization criteria, if applicable.

Further, requirements processes are expected to help establishing a meaningful concept of completeness as well as consistency. Yet, they must enable agile teams to take responsibility for their own ways of working. Recent works on boundary objects and bridging methodological gaps between different scopes in large-scale agile may offer useful guidelines.

5.6.6 Organization aspects

Related to the process aspects, our final challenge area includes challenges that relate to the overall organization in which requirements engineering is practiced. It is inherent to systems engineering that some long-term planning is needed, especially to plan for facilities to manufacture and test hardware and mechanics, but also to coordinate the integration of components across disciplines. Our challenges here relate to bridging between such system-level planning and agile work in software teams, to the planning of integrated system testing, to manage the research and pre-development, and to identify impacts on critical infrastructure in good time.

At the moment, we are not aware of proven approaches, neither through empirical evidence nor within agile frameworks, that can address these challenges. As with our challenges related to process aspects, we believe that recent research around boundary objects could offer a framework to encourage self-organization in system development. If constructively used to establish boundary objects as means of coordination between plan-driven an agile areas of an organization, we expect a positive impact on organizational aspects with engineering requirements in scaled-agile system development. Our works on T-Reqs can be seen as a special boundary object, where teams can communicate critical requirements changes early on and spread awareness through peer reviews.

5.6.7 Challenges beyond the scope of this study

Through the transition to large-scale agile, many aspects of the overall processes, organization and ways of working of our case companies were under consideration at the time of our investigation. Requirements are of critical importance to all of our case companies and they traditionally relate directly or indirectly to all aspects of system development. Thus, we found at several times...
during this investigation that we needed to sharpen the scope. We wanted to create a catalogue of general requirements-related challenges that are relevant to system development of organizations that have transitioned or desire to transition to large-scale agile.

One big challenge that we ultimately excluded from the scope relates to the development of safety-critical or regulated systems. It is an exciting research field, but deserves a dedicated space. Our challenges of large-scale agile system development also apply if safety-critical systems are developed, yet, safety and regulation bring in an additional level of complexity to an already complex topic. We will instead spend a few lines here to relate our findings to safety-critical systems.

Traditionally, long upfront analysis and planning aimed to address these needs [30]. However, as companies try to speed up their development, research needs to investigate new ways of dealing with documentation of such cross-cutting issues. Ensuring qualities and addressing non-functional requirements has been brought forward as a challenge in agile RE [15, 16], and first works exist to address regulations in agile [67, 68]. This is an interesting area, since it allows to look at requirements practice in large-scale agile as a spectrum, where regulation or safety demand for increased a more formal approach. Several of our case companies develop such systems and participants repeatedly expressed concerns that the development of safety critical software together with corresponding standards could impede agile development.

As examples, the participants expressed the need for documentation and tracing that is required by several standards, such as ISO26262 [69]. However, an expert for functional safety in Automotive 1 stated that the need for documentation and tracing is related more to the size of the company and the system rather than regulations.

"Many see that as a problem. Many say that it’s safety problem, it is a 26262 problem. But we say [...] we need to document anyway since then half a year later it is a different team [working on the same software]” — A1-TS

According to our interviewees, standard conformance could be combined with agile development if only this was planned in a systematic fashion, e.g., by sandboxing safety critical parts. Further, our case companies discussed a spectrum of requirements method ranging from full-scale for regulated and safety-critical systems to lightweight for unregulated and non-critical systems. Yet, it is unclear which concrete practices and approaches are distributed over this spectrum in large-scale agile, which is confirmed by our parallel work on safety in agile system development [176, 177].

5.7 Conclusion and Outlook

We presented our results from a multiple-case study with seven systems engineering companies on the interaction of RE and agile methods in large-scale development. We studied the pervasiveness of agile methods adoption, requirements-related challenges of large-scale agile systems development and solutions from best practices in industry as well as those provided by SAFe and LeSS. In all case companies, the way plan-driven and agile development
currently co-exist within the systems engineering environment limits the potential development speed. We found that in all companies, there is a need for strong requirements engineering approaches, especially with respect to documenting a system’s behavior for future feature requests or maintenance. The pervasiveness of agile implementation in the case companies differs, ranging from agile development on team-level embedded in an overall plan-driven process up to agile development for the entire product development. Despite the difference in pervasiveness, we observed similar challenges in all companies. These relate to establishing a shared view of value from the customer and other stakeholders down to development, supporting change and evolution, building up and maintaining a shared understanding about the system, representation of requirements knowledge, as well as dealing with process and organizational aspects. Proposals to mitigate these challenges have been extracted from SAFe and LeSS, and we have collected further practices from the companies. Despite these proposals and practices, we note that many challenges remain open or have solutions without realistic evaluation.

At the time of this investigation, we conclude that neither traditional requirements engineering nor scaled-agile frameworks provide satisfying concepts to manage requirements knowledge effectively, when developing at the scale and speed that our case companies desire. This encourages individual workarounds, for which we provide a comprehensive overview of challenges so that any solution does not over-optimize at the expense of a different challenge, and future research. In order to mitigate these challenges, we encourage future work to not only produce further practices to solve open challenges, but also focus on evaluation of existing large-scale agile proposals from a requirements perspective. Ideally, this will allow large-scale system development efforts to fully benefit from agile methods, while still systematically managing knowledge about customer value and the system under construction.

Acknowledgments

We thank all participants in this study for their great support, deep discussions, and clarifications. This work was supported by Software Center Project 27 on RE for Large-Scale Agile System Dev. and the Sida/BRIGHT project 317 under the Makerere-Swedish bilateral research program 2015-2020.
Table 5.6: Summary of results for Challenge Area 4: Representation of requirements knowledge.

<table>
<thead>
<tr>
<th>ID</th>
<th>Challenge</th>
<th>Proposed practices from case companies</th>
<th>Proposed practices from SAFe</th>
<th>Proposed practices from LeSS</th>
<th>Research gap in large-scale agile</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.a</td>
<td>Manage levels vs. decomposition</td>
<td>Allow stakeholders to specify on any level of abstraction, e.g. through traceability and reqts structure; Support distributed reqts analysis; Distinguish dimensioning FR and NFR</td>
<td>Clear hierarchy, from enterprise level to team; Requirements information model (epic-capability-feature-story): transport stakeholder view to components</td>
<td>Various splitting and refinement guides with depth 3 limit</td>
<td>Strategies and guidance for requirements decomposition, including how to manage customer and system requirements as well as on how to inter-relate them.</td>
</tr>
<tr>
<td>4.b</td>
<td>Quality requirements as thresholds</td>
<td>Thresholds to negotiate prizes; Trade-offs brought up by team and peer-reviewed by teams and system managers</td>
<td>NFR are constraints on program level, constraining a) the system and the product backlog or b) a feature and the team backlog</td>
<td>-</td>
<td>Strategies and guidance on managing and evolving quality requirements.</td>
</tr>
<tr>
<td>4.c</td>
<td>Tooling not fit for purpose</td>
<td>PO updates requirements; Team updates requirements</td>
<td>-</td>
<td>No software tools for sprint backlog Tools for large product backlogs (boards, pictures, wikis, spreadsheets?)</td>
<td>Tools specifically designed for large-scale agile practices, including reqts access control.</td>
</tr>
<tr>
<td>4.d</td>
<td>Accommodate different representations</td>
<td>-</td>
<td>Teams can have individual user stories flavour; emphasize team independence; Responsibility for Ways of Working, book clubs, guilds</td>
<td>-</td>
<td>Strategies and guidance to balance independence of teams and system level consistency.</td>
</tr>
<tr>
<td>4.e</td>
<td>Consistent requirements quality</td>
<td>Operationalization from experience; Peer-reviews by teams and system manager</td>
<td>Responsibility for Ways of Working, book clubs, guilds Community of practice to align on what is needed</td>
<td>-</td>
<td>Ways to share experiences on quality, empirical evaluation of suggested methods in practice</td>
</tr>
</tbody>
</table>
Table 5.7: Summary of results for Challenge Area 5: Process aspects.

<table>
<thead>
<tr>
<th>ID</th>
<th>Challenge</th>
<th>Proposed practices from case companies</th>
<th>Proposed practices from SAFe</th>
<th>Proposed practices from LeSS</th>
<th>Research gap in large-scale agile</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.a</td>
<td>Prioritization of distributed functionality</td>
<td>Business dashboard to help rank reqts; Clear product owner (hierarchy); More focus on interfaces, less on reqts; Estimation by risk</td>
<td>Combining &quot;weighted shortest job first&quot;, &quot;Portfolio backlog&quot;, and &quot;Program Kanban&quot; to support cross-cutting initiatives; Combine team backlog, business value, and interaction backlog; Sequencing based on cost of delay</td>
<td>One PO single source of prio.; Multi-site product backlog review; Challenge: Join the split-to-see problems</td>
<td>Empirical evidence and proven strategies on what works in specific context.</td>
</tr>
<tr>
<td>5.b</td>
<td>Manage completeness</td>
<td>-</td>
<td>No potential solution found in SAFe. Instead, build incrementally (Principle) and MVP suggest the opposite</td>
<td>LeSS-Guideline &quot;take a bite&quot;</td>
<td>Provide a clear taxonomy or language to reason about requirements completeness in incremental work at scale.</td>
</tr>
<tr>
<td>5.c</td>
<td>Consistent requirements processes</td>
<td>Delivery = code, test, and reqt (update)</td>
<td>Clear hierarchy, from enterprise level to team; Emphasis on team independence; Responsibility for WoW, book clubs, guilds</td>
<td>-</td>
<td>Strategies and guidelines to balance alignment and diversity of reqts practices.</td>
</tr>
<tr>
<td>5.d</td>
<td>Quality vs time-to-market</td>
<td>Frequent reviews of reqts in relation to Sprint deliverables</td>
<td>Clear hierarchy, from enterprise level to team; Built-in-quality; Reduce time-to-market; Value stream mapping</td>
<td>-</td>
<td>Guidelines to achieve just-enough quality of requirements, products, deliverables in order to reduce time-to-market.</td>
</tr>
</tbody>
</table>
### Table 5.8: Summary of results for Challenge Area 6: Organisational aspects.

<table>
<thead>
<tr>
<th>ID</th>
<th>Challenge</th>
<th>Proposed practices from case companies</th>
<th>Proposed practices from SAFe</th>
<th>Proposed practices from LeSS</th>
<th>Research gap in large-scale agile</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.a</td>
<td>Bridge plan-driven and agile</td>
<td>Dedicated governance of reqts across levels; Team updates requirements; PO updates requirements</td>
<td>Clear hierarchy, from enterprise level to team</td>
<td>-</td>
<td>Strategies and guidelines to replace static documents with actively managed boundary objects to allow coordination across levels.</td>
</tr>
<tr>
<td>6.b</td>
<td>Plan V&amp;V based on reqts</td>
<td>Establish virtual test rigs and simulation models; Manage reqts and tests together</td>
<td>Solution intent links specifications to tests; Duality of backlog items and tests; Cross-functional org.</td>
<td>-</td>
<td>Empirical evidence and proven approaches</td>
</tr>
<tr>
<td>6.c</td>
<td>Time for invention and planning</td>
<td></td>
<td>Solution intent, a repository of current and future solution behaviors Innovation and planning iterations Enabler stories</td>
<td>-</td>
<td>Empirical evidence and proven approaches</td>
</tr>
<tr>
<td>6.d</td>
<td>Impact on infrastructure</td>
<td>Establish virtual test rigs and simulation models</td>
<td>Feature teams Component teams Cross-functional org.</td>
<td>-</td>
<td>Proven strategies for achieving system-level awareness about critical reqt changes</td>
</tr>
</tbody>
</table>
Chapter 6

Paper E

Charting Coordination Needs in Large-Scale Agile Organisations with Boundary Objects and Methodological Islands

R. Kasauli, R. Wohlrab, E. Knauss, J.P. Steghöfer, J. Horkoff, S. Maro

*International Conference on Software and System Processes (IC-SSP ’20), 2020*
Abstract

Large-scale system development companies are increasingly adopting agile methods. While this adoption may improve lead-times, such companies need to balance two trade-offs: (i) the need to have a uniform, consistent development method on system level with the need for specialised methods for teams in different disciplines (e.g., hardware, software, mechanics, sales, support); (ii) the need for comprehensive documentation on system level with the need to have lightweight documentation enabling iterative and agile work. With specialised methods for teams, isolated teams work within larger ecosystems of plan-driven culture, i.e., teams become agile “islands”. At the boundaries, these teams share knowledge which needs to be managed well for a correct system to be developed. While it is useful to support diverse and specialised methods, it is important to understand which islands are repeatedly encountered, the reasons or factors triggering their existence, and how best to handle coordination between them. Based on a multiple case study, this work presents a catalogue of islands and the boundary objects between them. We believe this work will be beneficial to practitioners aiming to understand their ecosystems and researchers addressing communication and coordination challenges in large-scale development.
6.1 Introduction

Large-scale systems engineering companies are typically made of many teams that work together, commonly with plan-driven methods, to contribute to one product. With the recent digitisation trends, many such companies have adopted agile methods to help them address the ever-changing market needs and the growing competition\(^\text{44, 137}\). Given the challenges of introducing agility to large system companies\(^\text{13}\), most adoptions start with the software development teams at the lower level in the companies\(^\text{41}\). These teams in turn tailor the agile methods to their contexts\(^\text{84}\), thus creating companies that have pockets of agile teams within a larger ecosystem of plan-driven culture, also identified as “agile islands in a waterfall”\(^\text{138}\).

As the combination of agile with traditional plan-driven development methods become reality\(^\text{37}\), knowledge management and coordination challenges arise\(^\text{137}\). Inter-team coordination and knowledge sharing are key items on the research agenda on large-scale agile development\(^\text{45}\). Teams using varying methods and practices need to communicate to deliver the correct product. Artefacts shared at their boundaries, referred to as boundary objects, offer potential solutions to these knowledge and coordination challenges. Boundary objects have been proposed to help manage coordination between agile teams\(^\text{62}\). They can create a common understanding across sites, without compromising each team’s identity, and are therefore useful when establishing coordination mechanisms across team boundaries.

To this effect, and as a first step towards alleviating the coordination challenge, this study set out to understand and document the agile islands together with the boundary objects that are constantly encountered in large-scale systems development. Through an exploratory study with four large-scale system engineering companies, based on a focus group and two workshops, we explore the following research questions:

\textbf{RQ.1:} Which agile islands are repeatedly encountered in large-scale agile contexts? To understand how best to address the coordination issue, we uncover the different islands that are encountered in the companies.

\textbf{RQ.2:} Which boundary objects are repeatedly encountered in large-scale agile contexts? To understand how best they can coordinate and manage, we document the boundary objects that are shared between islands.

Interestingly, we find that not all islands reported are indeed \textit{agile islands} within a non-agile context. Significant distance can also occur between two agile teams and even the distance between two non-agile teams can have an impact on large-scale agile system development. In addition, we find concrete methodological islands, but also more abstract forces (or: drivers) that can contribute to the emergence of islands.

In answering our research questions, we provide a catalogue of methodological islands that are frequently relevant when introducing agility at scale, as well as the boundary objects between them. We believe that this study will benefit both researchers and practitioners who want to gain insights into inter-team coordination in large-scale development.

This paper is organised as follows: Section 6.2 presents the background to our study. In Section 6.3, we describe the methods we used to answer our questions and gives the details of the workshops and focus group. Section 6.4
presents our findings to RQ.1 and Section 6.5 describes the findings related to RQ.2. We discuss our findings and conclude the study in Section 6.6.

6.2 Background

Agile methods with the promise of continuous delivery of quality software have changed the way software is developed since the launch of the agile manifesto in 2001 [26]. Originally meant for small co-located teams, agile methods are being adopted in large-scale systems development organisations [178]. Existing studies on agile adoption in large-scale systems show that companies successfully adopt these methods [13, 12] even though challenges remain, especially those related to coordination and mixed processes between different teams [45, 179]. This section gives the background of agile islands and boundary objects presented in this study.

6.2.1 Agile Islands

Many large-scale system engineering companies have not fully adopted agile methods since they are not fully applicable in their domains. Empirical researchers have recommended tailoring agile methods to the contexts of the specific organisation [13]. This means that organisations have to carefully choose practices that complement their values, culture, and norms [180]. Research on agile methods tailoring has covered diverse areas including methods used to tailor [84] and also the rationales and implications of tailoring [153]. Still, to satisfy the need to complement agile methods with traditional methods, many companies are using hybrid methods in their development process [37].

Hybrid methods typically combine agile and plan-driven practices in software development [36]. Existing research on this topic has confirmed that this is the trend in many organisations today [37, 145]. Studies have explored the challenges faced in such environments [39, 137] and others have gone a step ahead to propose solutions [40]. Tell et al. [38] have studied how different practices are combined to devise hybrid processes in an attempt to understand how to systematically construct synergies.

It should be noted that in large-scale organisations in practice, combinations start with the software development teams using agile methods while the rest of the organisation works with traditional methods [41]. This leaves teams as “agile islands” in a waterfall environment [138], also defined as pockets of agile within larger ecosystems with plan-driven culture.

Vijayasarathy and Butler reason that the choice of method used in the teams is associated with characteristics of the organisation, project, and team size [181]. This offers an explanation for the existence of agile islands, that differ from the surrounding organization, e.g., in terms of artefacts, iteration length, and delivery schedule. Bjarnason et al. refer to such differences as different forms of distances, for instance, geographical, organizational, or cognitive distance, distance related to artefacts (e.g., semantic distance), and distance related to activities (e.g., temporal distance) [182]. Such distances makes it more difficult to coordinate between islands or between the non-agile part of the organisation and the agile islands.
6.2.2 Boundary Objects

Boundary objects are a sociological concept introduced by Star and Griesemer [64] who studied how a shared understanding between interdisciplinary stakeholders can be established. We refer to their definition of boundary objects as “objects which are both plastic enough to adapt to local needs and the constraints of the several parties employing them, yet robust enough to maintain a common identity across sites” [64]. In the context of agile development, the “parties [...] across sites” are individuals with potentially different backgrounds and disciplines, typically forming organisational units (e.g., teams or departments). These groups can flexibly interpret a boundary object and tailor it to their needs, while the group’s identity and existing practices can be preserved [183]. While boundary objects originate from the field of sociology, they have also been studied in agile development contexts (e.g., [184,185]). In these contexts, boundary objects are artefacts (e.g., design specifications or user stories) that create a common understanding between agile teams [185].

In large-scale agile systems engineering, boundary objects are used between individuals from several sub-disciplines of systems engineering, who refer to concepts with different terminologies and are often located at different geographic locations [62]. The groups using boundary objects need to be understood to enable knowledge management and inter-team coordination in an organisation. Some organisational groups might be agile islands, working in different ways than others parts of an organisation.

6.3 Research Method

Due to the exploratory nature of our research questions, we decided to conduct a multiple exploratory case study [186]. We collected data in a staged process, using a focus group with participants from several companies as a starting point and refining our data with in-depth workshops at two of the companies. Table 6.1 presents short descriptions of the participating companies. We report on our data collection, the way we analysed the information, as well as the threats to validity in the following.

6.3.1 Focus Group

We base the findings of this study on a focus group in which we discussed agile islands and the boundary objects that connect them with four practitioners, one from each of the four participating companies. Three of those practitioners had prepared presentations based on our instructions to help us explore the following issues: (i) in terms of inventory, what knowledge is required on the island and what knowledge actually exists; (ii) in terms of infrastructure, what knowledge needs to be shared and what knowledge is actually shared; and (iii) in terms of process, how to facilitate learning, retrieving, capturing and applying knowledge.

The practitioners have high-level technical roles in the organisation (system architect, tooling and process specialist) and are thus accustomed to working with different islands within the organisation. They have also been working in these companies for several years and thus have a good grasp of the processes
6.3. RESEARCH METHOD

Table 6.1: Descriptions of participating companies

<table>
<thead>
<tr>
<th>Focus Group</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Company A</strong></td>
</tr>
<tr>
<td><strong>Company B</strong></td>
</tr>
<tr>
<td><strong>Company C</strong></td>
</tr>
<tr>
<td><strong>Company D</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>First Workshop (Company B)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>13 practitioners</strong> Systems engineers, project managers, test specialists, digital transformation managers, and business developers.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Second Workshop (Company A)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>3 practitioners</strong> Scrum master, architect, systems engineer</td>
</tr>
</tbody>
</table>

and the organisational structure. All four companies are large-scale systems engineering organisations with a predominantly agile software development approach and global distribution of developers.

The three presentations identified boundary objects commonly encountered in practice together with teams that use them. They provided a foundation for identifying common boundary objects and islands and were the starting point for discussions about the commonalities and differences between the organisations.

This information was collected by the researchers with extensive notes. One of the researchers also prepared an overview image of the boundary objects and the islands they connect and applied a rough clustering while the workshop was ongoing. This figure was continuously augmented with new insights during the presentations and updated during the discussion. At the conclusion of the focus group, the figure was presented and practitioners could comment on whether it represented their understanding.
6.3.2 Individual Company Workshops

As a follow-up of the focus group, we conducted individual workshops of approximately three hours each with two of the companies that participated in the initial data collection. The workshops were conducted onsite at the companies and aimed to analyze concrete agile islands and boundary objects based on the inventory from the focus group. Two researchers were involved in each of the workshops and acted as moderators and facilitators. We prepared a workshop instrument (https://rebrand.ly/workshop_BOMI) to introduce the topic and guide through the workshops.

The workshops started with an introduction to boundary objects and agile islands and a statement of the goals. The participants were then asked to individually brainstorm the boundary objects and agile islands they encounter in their work. All input was recorded on post-it notes. Agile islands were then discussed and roughly organized on a wall. Once a picture of the relevant islands emerged, participants then located boundary objects between the identified islands creating a map. This map was then discussed and the practitioners reflected on the implications of the islands and how the boundary objects are currently being managed.

The first company workshop attracted a total of 13 practitioners who represented a number of roles: systems engineers, project managers, test specialists, digital transformation managers, and business developers. Representatives from the company first presented their current development process and the transformation that they are undergoing. Afterwards, the two researchers introduced agile islands and boundary objects and defined the purpose of the workshop. We then followed the procedure outlined above. However, after the collection of islands and boundary objects and the initial discussion of the map, we focused on a specific boundary object ("Product Requirement Specification") that was deemed highly critical by the practitioners. This provided additional insights into differences of governance processes within the organisation as well as the impact of organisational cultures in different parts of the company.

At the second workshop, three company participants attended, having the roles of Scrum master, architect, and systems engineer. The workshop procedure outlined above was followed, starting with an introduction of the concepts and goals, and ending with reflections on the implications of the findings. The focus lay on Interface Descriptions, Product Backlog, and Customer Service Requests, and collected relevant characteristics for them.

In each workshop, two researchers took detailed notes of what was being said as well as pictures of the post-it notes. Directly after the meeting, reflections were written down to allow for easier analysis.

6.3.3 Data Analysis

All collected data was discussed between the researchers in groups. We used coding [187] to identify common themes in the agile islands and boundary objects we collected and structure the information in our transcripts and notes, as well as in the documents we collected from the practitioners. Discussions continued until an agreement about the codes was reached within the group of researchers. All findings were then member checked [188] with the practitioners.
from whom the data was collected. The final results of these efforts provide the answers to the research questions outlined in Section 6.1 and are presented in the following. We collected the majority of our boundary objects and islands in the focus group. The follow-up company workshops confirmed the existence of many of these islands and boundary objects, adding only a few new elements, increasing our confidence in our findings as per this set of companies. We demonstrate this process by including our initial overview after the focus group (Fig. 6.1), a sample picture from the whiteboard after brainstorming with Company A (Fig. 6.2), and a mindmap with the first draft of results reported in this paper (Fig. 6.3). In Figure 6.3, it can be seen that our initial findings were classified as boundary objects and islands, as well as “technological drivers”, “process drivers”, and “organisational drivers”. These findings were refined in several steps to arrive at the final results reported in this paper.
6.3.4 Threats to Validity

We addressed threats to *internal validity* by including a number of practitioners in our workshops whenever possible and by allowing them to discuss their different perspectives on the data we collected. This increases our confidence that the data which forms the foundation of our study corresponds to the reality at the organisations that participated in it. The positive outcome of member checking our results further compounds this.

In terms of *external validity*, we do not claim that our findings in terms of the concrete methodological islands and boundary objects we found are complete. By analysing data gathered from different companies with different characteristics, however, we believe that we have sketched out a framework that can be extended in the future and were able to identify relevant categories that are applicable in other contexts. It is our intention to extend the catalogue presented here and create a conceptual model of Boundary Objects and Methodological Islands (BOMI) with higher generalisability in the future.

To mitigate threats to *construct validity*, we began each workshop with presentations explaining the concept of boundary objects and provided examples to help understanding. The focus group targeted high-level experts from the respective companies. As reflected in their presentations, these experts understood the concepts well. Also, questions were asked and clarifications made throughout the workshops. Thus, all of our data collection tools focused on improving the understanding of the constructs under investigation, i.e., boundary objects and methodological islands. Evaluation apprehension or experimenter expectancies are potential threats to construct validity. Peer debriefing helped us to critically reflect on these potential factors and the impact on our findings. To address *reliability*, we combine a focus group with individual workshops at companies and combine the data collected in both to derive overall findings.
6.4 Frequently Encountered Agile Islands

In this section, we present our findings with respect to RQ.1 (Which agile islands are repeatedly encountered in large-scale agile contexts?). Overall, the discussion of agile islands resonated very well with our industry participants, both in cross-company workshops and in focus groups with individual companies. When analysing the collected data, we found a wide spectrum of relevant islands that we had to organize and categorize. This led us to two observations: (i) not all islands that were mentioned are in fact agile islands. Thus, there can be significant distance between two agile teams and even the distance between two non-agile teams can have an impact on large-scale agile system development. For this reason, we started to refer to the islands as methodological islands. (ii) not all islands mentioned were on the same level of abstraction. While some (e.g., individual teams) are very concrete, others (e.g., “software vs physical components”) are not very concrete agile islands, but can be seen as contextual factors that cause islands to emerge. We therefore started to refer to the latter as drivers of methodological islands.

We first start to describe concrete methodological islands, before we also share the abstracted drivers.

6.4.1 Methodological Islands

The islands derived occur on different levels in the organisations. In Table 6.2 we give an overview of levels and typical examples of islands.

6.4.1.1 Groups of teams

Two of our participants’ companies implemented the Scaled Agile Framework (SAFe). SAFe suggests the use of Agile Release Trains, i.e., of a team of agile teams that together develop and deliver a solution. Value streams exist on the highest level of SAFe. Within each value stream, there are multiple release trains. In one of the participating companies, there are about 50 release trains with 5 to 12 teams in total. Internally, these agile release trains require synchronisation and coordination, but externally, they can be perceived as a black box. These release trains develop different (sub-)systems that have interfaces with each other. When several release trains depend on each other, their differences in methodology become an obstacle. Officially, departments are not mentioned anymore in the SAFe-related documentation, but have traditionally existed in the companies. Release trains are orthogonal organisations to the former departmental structures and can span several departments in the company.

Product development typically spans several departments in an organisation. These departments, for example, marketing, hardware development, embedded system development, come from different contexts and thus different ways of working. As it was not yet clear how hardware can work in an agile way or if they even should, the hardware teams for instance maintained plan-driven methods and yet they have to interface with software teams that are already adopting agile methods. Hardware and software departments work using different timelines. It is also common for globally distributed companies to have departments spanning different locations that could spur different methods within the department due to the difference in cultures. Each department can
### Table 6.2: Examples of Methodologicical Islands on different levels.

<table>
<thead>
<tr>
<th>Groups of teams</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Agile teams</td>
<td>Agile release trains or scrum-of-scum clusters may differ in methods.</td>
</tr>
<tr>
<td>Departments</td>
<td>Different methods and practices for SW development have emerged in different</td>
</tr>
<tr>
<td></td>
<td>departments of large system companies (e.g., infotainment, powertrain in</td>
</tr>
<tr>
<td></td>
<td>automotive)</td>
</tr>
<tr>
<td>Disciplines</td>
<td>Systems engineering needs to combine several disciplines, including hardware,</td>
</tr>
<tr>
<td></td>
<td>mechanics, and software of different types, each with their own set of methods and</td>
</tr>
<tr>
<td></td>
<td>practices.</td>
</tr>
</tbody>
</table>

### Individual teams

| Component teams          | If teams are related to architectural components, they may favour different       |
|                         | methods and practices.                                                           |
| Integration teams       | Complex products may require dedicated support for continuous integration,       |
|                         | provided by specialized testing and framework support teams. Their methods may    |
|                         | differ significantly from other teams.                                            |

### Organizations

| Suppliers                | If an OEM aims for continuous integration, they may require suppliers to         |
|                         | continuously deliver SW components. Naturally, methods and practices differ      |
|                         | between customer and supplier and between suppliers.                            |
| Consultants             | Systems engineering companies may rely on consultants to help developing software |
|                         | components. These may again bring a different set of methods and practices.     |
| Regulators              | Agile system development of regulated systems needs to take into account methods  |
|                         | and practices of regulators. These may differ between domains and particular     |
|                         | regulators.                                                                      |

Have several teams with a common goal of contributing to a single component or feature of the product.

### 6.4.1.2 Teams (individual teams)

Within an organisation, different teams can follow different agile approaches or even work in an agile way while the rest of the organisation follows a plan-driven approach. Teams in such large companies handle different parts of the architecture of the product. This means that each team works with different requirements and thus could use different approaches to get to the solution. Participants mentioned, for instance, continuous Integration framework teams, integration testing teams, Web GUI teams, and software teams. All of these teams may contribute to the same product, but since the nature of their tasks differs significantly, they often tailor development processes to their needs. This leads to a set of methodological islands throughout the organization.
6.4.1.3 Organizations

Companies work with suppliers, customers and regulators all of which come with different ways of working from that of the corresponding company. The suppliers provide some components while others are developed in-house. When teams within an organisation rely on external suppliers for components, the supplier is often working in a waterfall way. For instance, contracts between both companies often imply a plan-driven approach since purchasing is based on clearly defined functionality to be delivered at a certain point in time. Regulators also rely on standards that do not explicitly specify the methods to use in development, but come with checkpoints that relate mostly to the plan-driven methods of working. This mismatch of the actual methods used versus the ‘unknown’ expectations becomes a hindrance in development.

6.4.2 Drivers of Methodological Islands

The methodological islands are triggered by certain factors that we derived upon analysis. We summarize these in Table 6.3 and describe them below.

Table 6.3: Different types of drivers for methodological islands.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business-related</td>
<td>Economic function, Characteristics of market or value-chain, global distribution</td>
</tr>
<tr>
<td>Process-related</td>
<td>Mixture of development methods (SAFe, V-Model, Scrum, Kanban, LSD); focus on projects or products</td>
</tr>
<tr>
<td>Technology-related</td>
<td>Architectural decomposition, systems disciplines, platform and product-line strategy, time-scale of commitment</td>
</tr>
</tbody>
</table>

6.4.2.1 Business-related drivers

Based on their history and business domain, companies have specialized organisational charts that describe the departments, e.g., for marketing, development, verification. These departments handle different parts of the product that in most cases imply varying needs for development methods. For instance, the sales department as opposed to development departments, have different needs and thus different ways of working. This difference, in turn sparks the need to adopt the agile practices to the context of the specific department, causing islands of methodology, for example, manifesting in different choices with respect to forming cross-functional component or feature teams. Such business drivers can be the result of a particular culture in a market or value-chain. How is the relationship between customers and suppliers characterised in terms of contracts, time-lines, trust, and interaction? To what extent are customers willing or able to assess and give feedback on frequent deliveries? Is it possible to take end-user opinions into account and to what extent do they matter? All these aspects contribute to the mix of methods and how the other stakeholders are going to work to get the product they need.

Some of our participant companies are distributed over several countries and in some cases different areas in the same country. Developers of software or hardware do not work in the same buildings and are separated by location,
time zones and culture. This separation in the end creates teams that have defined different methods of doing the same thing.

6.4.2.2 Process-related drivers

While teams exist in the organisation and have varying needs, the organisation in the end has to have one backbone process that defines the company. It is not uncommon that different teams within the same organisation use different flavours of agile methods. Apart from process customisation for each individual team, it is possible that some teams use a method such as SAFe while others employ Scrum, Kanban, XP, or a form of lean development. These differences introduce islands where roles, artefacts, and schedules are difficult to coordinate.

A major driver of this category relates to whether a company mainly works based on projects, or whether significant work flows in the continuous development of a platform. While projects are adding features to a solution and are thus short-term, platforms are planned for the long-term. Platforms need to be more stable since other projects depend on them and changes in the platform can have a major impact on the depending projects. Thus, the particular setup of a company can create islands between different projects, or between customer projects and platform development.

6.4.2.3 Technology drivers

Complex systems are often developed by different teams that are responsible for individual parts of the architecture. That means that these teams not only address different sets of requirements, but also apply different technologies in their solutions. Teams working on software and on physical components work according to different timelines and according to different cultures. Hardware development often assumes stable requirements and development of a full solution, instead of development of slices of functionality and rapid response to changes.

Many companies with complex product lines, e.g., in the automotive domain, produce platforms as the foundation of their products. Platforms are often generational, i.e., they are used for a certain period of time before they are replaced by the next generation. Each platform has a unique technical solution and is usually not compatible with previous ones. At the same time, different teams working on different platform generations also often use different generation of processes.

The time scale of commitment is another technology-related driver. Agile methods usually imply short-term commitment in individual sprints. That means that requirements can change from sprint to sprint to react to a changing market situation or newly discovered opportunities. On the other hand, many methods require a longer-term commitment. Platforms, e.g., that are used by many other projects and thus need to be stable might be better served using a plan-driven approach and to constitute “waterfall islands” within the organisation.
## 6.5 Boundary Objects in Large-Scale Agile

In this section, we answer **RQ2: Which boundary objects are repeatedly encountered in large-scale agile contexts?**

Table 6.4: Identified boundary objects and their categories

<table>
<thead>
<tr>
<th>Task Boundary Objects</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Backlog item</strong></td>
<td>Backlog items, e.g., from a product backlog, can be representations of high-level requirements and are used by individual islands to define their own, local backlog items for their product or team backlogs.</td>
</tr>
<tr>
<td><strong>(User) story</strong></td>
<td>A user story is an upfront feature description focused on customer value. Backlog items can be formulated as user stories to clarify the value provided by delivering a piece of functionality.</td>
</tr>
<tr>
<td><strong>Feature, function description, or high-level requirements</strong></td>
<td>A set of high-level requirements can be represented by a feature or function description. These requirements need to be further broken down to allow individual islands to work on them.</td>
</tr>
<tr>
<td><strong>Requirements on interfaces</strong></td>
<td>Different parts of a software architecture are connected by interfaces. The requirements for these interfaces define contracts between teams. For instance, timing requirements on an interface need to be adhered to by all islands using this interface.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Technology Boundary Objects</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Capabilities</strong></td>
<td>A description of the capabilities provided by the system gives a high-level overview of the functionality. It allows individual teams to identify relevant reusable assets and required interfaces.</td>
</tr>
<tr>
<td><strong>Automated tests</strong></td>
<td>Integration, acceptance, and non-functional tests can be shared between islands to avoid regressions, ensure customer value is jointly achieved, and to document the functionality provided in the system.</td>
</tr>
<tr>
<td><strong>API / Interface</strong></td>
<td>The description of the interfaces between different parts of the solutions allow to modularise the development and different islands to reuse existing assets.</td>
</tr>
<tr>
<td><strong>Reference architecture</strong></td>
<td>A high-level description of the architecture both allows different islands to identify where a feature should be located and ensures that new additions to the solution follow the common guidelines of the organisation.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Regulation and Standards Boundary Objects</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standards</strong></td>
<td>Safety standards such as ISO 26262, DO-178B, or IEC 62304, prescribe development practices and artefacts.</td>
</tr>
<tr>
<td><strong>Regulations</strong></td>
<td>Regulations take the role of standards and prescribe certain practices or artefacts (e.g., in the telecommunications domain).</td>
</tr>
<tr>
<td><strong>Safety assurance case</strong></td>
<td>Safety standards prescribe the creation of white box or black box safety assurance cases that describe how a product addresses risks during its operation. These cases can be used by different islands to understand the risks involved in the system and to develop common strategies to avoid them or deal with them.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product Description Boundary Objects</th>
<th></th>
</tr>
</thead>
</table>
Table 6.4 – continued from previous page

<table>
<thead>
<tr>
<th>Variability model</th>
<th>The features of a product and the constraints between them (e.g., which ones are mutually exclusive or incompatible) can be used by different islands to understand the interaction between their solutions and the rest of the product line.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical documentation for customer</td>
<td>Outwards-facing documentation can also be used internally to gain a common understanding of how different parts of a system are related.</td>
</tr>
</tbody>
</table>

**Process Boundary Objects**

| SAFe documentation | The Scaled Agile Framework (SAFe) has found widespread adoption in large development organisations. It provides detailed documentation and support for its adoption. This documentation can, together with a description of how SAFe was adapted, act as a boundary object between islands using SAFe and other parts of the organisation not using SAFe. |

**Planning Boundary Objects**

<table>
<thead>
<tr>
<th>Contracts</th>
<th>The interactions between parts of an organisation and the suppliers are often defined by contracts. Contracts can also bind an island within an organisation to external constraints. In any case, the content of the contract will define the scope or the time and resources the island has at its disposal.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roadmaps</td>
<td>The long-term evolution of a product is often defined by one or several roadmaps. These boundary objects also link different products that co-evolve to each other. Therefore, they are used to coordinate between islands within an organisation.</td>
</tr>
<tr>
<td>Short-term plans</td>
<td>The development of individual features or smaller parts of a product is often bound to a shorter-term plan that is connected to the overall, long-term plan. As such, the scope of a short-term plan is also limited to a smaller number of islands.</td>
</tr>
<tr>
<td>Resource budgets</td>
<td>When developing systems in which software runs on dedicated hardware, individual islands need to work with a resource budget that determines how much computing power, memory, or bandwidth their specific functionality can consume.</td>
</tr>
<tr>
<td>Trace links</td>
<td>Artefacts created during development need to be connected to each other using trace links. They clarify the relationship between artefacts and enable change impact analysis and collaboration between the islands that created the artefacts.</td>
</tr>
</tbody>
</table>

Table 6.4 shows our findings for RQ2. Each row represents a type of boundary object with a description of how it facilitates coordination between islands. In total, 19 types of boundary objects were identified. We categorized them in different themes: Task, technology, regulation and standards, product description, process, planning boundary objects, and trace links. We refer to the descriptions of the boundary objects in Table 6.4 and briefly summarize the categories in the following.

Task boundary objects relate to tasks in the development effort in which boundary objects facilitate the creation of a common understanding across team borders. Concretely, these tasks are concerned with identifying development activities by creating a backlog and specifying requirements to define the
Figure 6.4: Summary of findings: Certain drivers introduce distance, which in turn frequently introduces methodological islands on different levels. Boundary objects of various types can be crucial to bridge between islands and support effective agile system development at scale.

functionality to be developed. Typical examples are user stories and other backlog items as well as related comments stored in issue trackers.

*Technology boundary objects* are concerned with technological aspects of the (software) system to be developed, including a system’s capabilities, tests, or architecture boundary objects. These boundary objects are commonly used between different islands and mostly by technical stakeholders.

*Regulation and standards boundary objects* are used to ensure that the company complies with regulations and standards. In our case companies, it relates mostly to safety standards, as with the safety assurance case. These regulations and standards are typically relevant across island borders and a common understanding of these concerns is required.

*Product description boundary objects* relate to the product as it will be sold to the customer. While the respective documentation is mostly created for customers, it can also be leveraged internally, for instance, to create a shared understanding of variability concerns or other technical aspects.

*Process boundary objects* are concerned with documentation regarding processes or frameworks. In two of our case companies, SAFe is used and tailored to each company’s needs. The created documentation can help to get a shared understanding of the processes and roles.

*Planning boundary objects* relate to contracts, roadmaps, plans, or budgets that are created before development. These boundary objects are commonly used between non-technical stakeholders like managers, but can also be relevant for development teams.

*Trace links* are a special category, as they represent the relationships between artefacts. Trace links typically have types that determine how other artefacts relate to each other. They can also serve as boundary objects between different methodological islands, capturing a mutual agreement about relatedness of other boundary objects.
6.6 Discussion and Conclusion

In this paper, we presented methodological islands and boundary objects related to large-scale systems development collected from two workshops and a focus group with four large organisations. We present a summary of our findings in Figure 6.4. As the first step towards addressing the coordination challenge in transforming organisations, we believe that this study adds significant value both to research and to other organisations customising agile. We discuss our main findings and implications of our work in this section.

6.6.1 Methodological Islands

We discuss RQ.1 (What are possible scopes of applying agile methods in large-scale system development?) in this section.

Our findings show that when embracing agile in large-scale system development, certain types of methodological islands frequently appear on the level of individual teams, groups of teams, or full organisations. Although not particular to large-scale, West et al. [41] found that water-scrum-fall is becoming a reality for most organisations, a claim confirmed by Theocharis et al. [37]. While terming them hybrid methods, Kuhrmann et al. [36] find that such hybrid approaches are not limited to traditional and agile development but also allow combinations of different agile methods since agile is also not implemented as is. Tell et al. [38] go a step further and identify the agile methods and how they are combined in practice to form hybrid methods. Our findings on methodological islands confirm their findings as well as recognising that such combinations differ within the same organisation, causing methodological islands. Such islands are characterised by their relative distance in terms of methods and practices as well as culture and mindset.

In addition to the methodological islands, we found that certain drivers (business-, process-, and technology-related) can introduce such distance and lead to the formation of methodological islands. This finding concurs, to some extent, with the finding by Vijayasarathy and Butler [181] who found specific organisational, project and team characteristics had an effect on the choice of methodology. While we can confirm several of those characteristics, we come from the perspective of islands and classify the drivers as business-related, technology-related and process-related. Team characteristics could play a role but for the islands context, these are overtaken by e.g., the nature of (sub-)systems that different teams may be developing and thus we relate that driver to technology.

6.6.2 Boundary Objects

In this section, we discuss the findings of RQ2: How is the role of requirements characterized in large-scale agile system development?

In order to successfully introduce agile methods and to deliver a full product or system, we found that effectively bridging between such islands is crucial. We believe that it is beneficial to think about artefacts that support such bridging as boundary objects and provide in this paper an inventory of frequently encountered boundary objects. Many of the identified boundary objects have been confirmed by related studies. In an analysis of boundary objects in distributed
agile teams including developers and user-centered design specialists, Blomkvist et al. identified the following boundary objects: (1) Personas, (2) Scenarios, (3) Effect maps, (4) Sketches, (5) Design Specifications, (6) Prototypes, (7) Evaluation summaries, and (8) User stories [185].

Our findings include a System Wiki boundary object, identified by company A. Similarly, Yang et al. [189] name the use of a wiki as a boundary object for requirements engineering. The accessibility and ability to simultaneously access and create information make wikis a suitable form for a boundary object.

In an analysis of boundary-spanning activities with a focus on requirements engineering practices for product families, examples boundary objects included traceability documentation, process models, vocabularies, user stories, product/process repositories, XP practices, feature models, the product backlog, the sprint backlog, and product prototypes [190].

In the area of requirements engineering, another study has focused on classification schemes as boundary objects, allowing stakeholders to categorise requirements in different ways (main users, functional vs. non-functional, level of abstraction) [191]. In fact, standardised forms and classification schemes have been examined in the context of boundary objects since their initial definition [64, 192]. The regulation and standards boundary objects that we identified in this study relate to this category. Moreover, process boundary objects potentially include classification schemes, for instance, by defining requirements information models that determine how stakeholders should work with requirements-related concerns and how they should be categorised [170].

Focusing on software development, project management documents and specifications have also been identified as boundary objects [193]. Thus our findings confirm many existing objects, and create a more integrated, industry-driven view of such objects in a large-scale agile context.

6.6.3 Implications for practitioners

We found our inventory of methodological islands and related boundary objects useful when discussing potential process improvements with companies. Already the focus group and company workshops showed that this facilitates a useful mindset, where artefacts are discussed as a means to satisfy coordination needs between methodological islands. By making the islands explicit and by discussing their particular context, mindset, and preference with respect to methods and practices, we believe that such boundary objects can be established in a better way than if they would emerge in an unplanned way, e.g., by re-using non-agile artefacts. Future work should investigate if this can be used constructively, when defining or improving processes, methods, and tools.

6.6.4 Implications for research

Similarly, we hope that charting the landscape of methodological islands and boundary objects in large-scale agile system development will create a useful model to scope and prioritize future research. Future research could refine the classifications in our inventory and provide a suitable conceptual model or taxonomy. We are currently working with our participating companies to
derive possible recommendations and best practices for boundary objects based on their properties. In addition, a quantitative study could provide additional information on which boundary objects and methodological islands are most frequent.

**Acknowledgement**

This work is partially funded by Software Center, Project #27 “RE for Large-Scale Agile System Development” [www.software-center.se](http://www.software-center.se) and the Sida/BRIGHT project 317 under the Makerere-Sida bilateral research programme 2015-2020. We thank our industry partners for their enthusiasm and support.
Chapter 7

Paper F

Note: This chapter includes a paper published at RE’18, Tool Demonstration Track and combines it with a technical report, which offers more detail on how T-Reqs relates to challenges revealed in this PhD thesis.

T-Reqs: Tool Support for Managing Requirements in Large-Scale Agile System Development

E. Knauss, G. Liebel, J. Horkoff, R. Wohlrab, R. Kasauli, F. Lange, P. Gildert

Proceedings of 26th International Requirements Engineering Conference (RE’18), 2018

A Simplified Guide to the use of T-Reqs

R. Kasauli and E. Knauss

Technical report, arXiv, 2020
Abstract

T-Reqs is a text-based requirements management solution based on the git version control system. It combines useful conventions, templates and helper scripts with powerful existing solutions from the git ecosystem and provides a working solution to address some known requirements engineering challenges in large-scale agile system development. Specifically, it allows agile cross-functional teams to be aware of requirements at system level and enables them to efficiently propose updates to those requirements. Based on our experience with T-Reqs, we i) relate known requirements challenges of large-scale agile system development to tool support; ii) list key requirements for tooling in such a context; and iii) propose concrete solutions for challenges.
7.1 Introduction

Requirements engineering (RE) is crucial to support agile development of large systems with long lifetimes. Yet, traditional tooling does not sufficiently support addressing known RE challenges in large-scale agile \([15, 16, 23]\). In particular, it is hard to cross the boundaries between three domains in large-scale agile organizations: the customer facing domain, the development domain, and the system domain \([138]\). Agile cross-functional teams must be aware of requirements at system level and able to efficiently propose updates to those requirements. Based on real world experience with selecting, rolling out, and, for several releases now, using a novel tool solution based on git and markdown, we i) relate known requirements challenges of large-scale agile system development to tool support; ii) list key requirements for tooling in such a context; and iii) propose concrete solutions for challenges.

7.2 Industrial Context

We report here based on the experience from one particular product developed at a specific department within Ericsson. The product development organization of the department has changed significantly in recent years. Before 2012, the development process defined two releases per year, requirements were created upfront, and were defined in IBM Rational Requisite Pro (first released in 1995). In 2012, the department started a transformation towards agile methods and continuous integration at scale. In 2017, this transformation has reached a state where the product is released once per month.

While it is hard to quantify the level of agility in the company, the changes are significant and comprehensive. The development process is based on a continuous feature flow and allows about 30 small, cross-functional development teams (XFT) to develop software following a Scrum approach. This has significantly changed the way of managing requirements.

The following three changes stick out:

| C1 | Requirements are updated more frequently. This can cause teams to block each other. |
| C2 | Number of database clones grows. The need to clean up databases because of unfinished requirements as well as to frequently port requirements back to a main database causes significant effort. |
| C3 | No direct link between requirements and commits. It is difficult to determine when a feature has been delivered. |

The existing requirements tool was no longer deemed sufficient and the department evaluated 16 different requirements engineering tools. None was found to satisfy their specific needs. Two requirements stuck out as especially hard to fulfill:

| R1 | The tool must use git as version control system or must support easy synchronisation with git. Teams are working with git and since it is their responsibility to propagate changes to requirements as part of a sprint, a suitable requirements tool must allow so within the development context. |
| R2 | The tool must support simultaneous work of many users on the same artefact. Since several teams may be working in the scope of particular requirements, they must be able to report their changes without reserving the artefact and blocking each other. |
When searching for a suitable tool, an in-house proposal proved to provide the best fit: T-Reflections. T-Reflections (textual requirements) suggests managing requirements in markdown format in text files within git. As depicted in Table 7.1, this surprisingly simple solution satisfies the majority of concerns and fulfils the critical requirements with only small additions (scripts and templates) beyond existing solutions in the git ecosystem. These allow for example to generate reports and views on the requirements and related artifacts with tracing links.

### 7.3 T-Reflections vs. Challenges and Requirements

Table 7.1: Challenges and requirements for requirements tooling in large-scale agile system development.

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Current status</th>
<th>User story/requirement</th>
<th>Solution in T-Reflections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updating and deprecating requirements.</td>
<td>Updates of requirements must be proposed to a central role. The process is slow; the central role becomes a bottleneck, and changes that appear non-critical may be omitted ($\rightarrow C1$).</td>
<td>US1 As a member of a XFT, I want to a) ... share new knowledge we learnt about existing requirements during a sprint so that our implementation and the requirements on system level are consistent. b) ... be aware of requirements changes that affect my team so that we can pro-actively address dependencies.</td>
<td>Git allows to group changes (e.g. to source code and tests) into commits that then can be pushed to the main branch. T-Reflections allows to manage requirements in exactly the same way. A team pulling the latest changes from the main branch will see conflicts on either of these artefacts as merge conflicts in git.</td>
</tr>
</tbody>
</table>

Continued on next page

---

1 Demonstrator available at [https://github.com/regot-chalmers/treqs](https://github.com/regot-chalmers/treqs)
Table 7.1 – continued from previous page

| Access to tooling and requirements. | Often, teams do not have access to tooling and requirements, since licenses are expensive and the number of parallel users is limited ($\rightarrow C1$). | US2 As a member of a XFT, I want to update system requirements efficiently, without too much overhead, and ideally integrated in the tools I use in my daily work. | Development teams are used to git, thus T-Reqs provides them with a familiar interface to manipulate requirements. T-Reqs-specific conventions, templates and scripts allow generating specific views and reports for non-technical stakeholders. |
| Consistent requirements quality. | No appropriate review and alignment process exists that would allow to include an individual team’s way of working ($\rightarrow C1-2$). | US3 As a system manager, I want to make sure that proposed updates to requirements are of good quality, do not conflict with each other, or with the product mission. | Many organisations that rely on git are also using gerrit to manage reviews (e.g. of source code). T-Reqs organizes requirements in a way that allows to do that with requirements as well. |
| Managing experimental requirements. | The requirements database is cloned before experimenting and must be blocked for other changes when the clone is eventually be ported back ($\rightarrow C2$). | US4 As member of an experimenting team, I want to experiment with new requirements and features so that I can better assess their business value and cost. This must not affect existing requirements during the experiment or block the requirements database afterwards. | Git allows creating branches to experiment with requirements, but also with models and source code. Git merge and gerrit help to merge branches, without blocking the main database. Merge conflicts will directly relate to requirements conflicts, since requirements are stored line-wise. |

| Consistent requirements quality. | No appropriate review and alignment process exists that would allow to include an individual team’s way of working ($\rightarrow C1-2$). | US3 As a system manager, I want to make sure that proposed updates to requirements are of good quality, do not conflict with each other, or with the product mission. | Many organisations that rely on git are also using gerrit to manage reviews (e.g. of source code). T-Reqs organizes requirements in a way that allows to do that with requirements as well. |

| Managing experimental requirements. | The requirements database is cloned before experimenting and must be blocked for other changes when the clone is eventually be ported back ($\rightarrow C2$). | US4 As member of an experimenting team, I want to experiment with new requirements and features so that I can better assess their business value and cost. This must not affect existing requirements during the experiment or block the requirements database afterwards. | Git allows creating branches to experiment with requirements, but also with models and source code. Git merge and gerrit help to merge branches, without blocking the main database. Merge conflicts will directly relate to requirements conflicts, since requirements are stored line-wise. |
Table 7.1 – continued from previous page

<table>
<thead>
<tr>
<th>Create and maintain traces.</th>
<th>Tracing does not offer direct value to agile teams and is not integrated in their workflows ($\rightarrow C3$).</th>
<th>US5 As a member of a XFT, I want to maintain traces between requirements, change sets, and tests in a way that is integrated with my natural workflow and enables valuable feedback.</th>
</tr>
</thead>
</table>
| In continuous integration and delivery, agile teams struggle to provide sufficient tracing to allow determining the status of individual features. | Git automatically links changes of code and requirements in commits. T-Reqs adds conventions, templates, and scripts for additional finer-grained tracing. Providing cross-functional teams with feedback based on tracing information can further motivate good trace link quality. |}

Plan verification and validation based on requirements. When requirements changes are difficult to share and the need to update complex system testing infrastructure may surface late.

| Requirements changes are difficult to share ($\rightarrow C1.3$) and the need to update complex system testing infrastructure early on so that I can plan verification and validation pro-actively. |
| US6 As a test architect or system manager, I want to be aware of new requirements for the test infrastructure early on so that I can plan verification and validation pro-actively. |
| T-Reqs suggests a suitable review process via gerrit that has proven to spread information about critical changes effectively between key stakeholders. Ease of use makes it more likely that teams share requirements updates in a timely manner. |

Recently, a growing number of empirical papers discusses requirements engineering challenges in agile development [15, 16, 138] as well as the need to identify new ways of managing requirements in agile organizations [23]. In Table 7.1, we select a subset of these challenges that is especially relevant to tooling in our company context. We also extract requirements towards tooling in form of user stories and discuss how T-Reqs helps to address these challenges.

### 7.4 Discussion and Outlook

In this paper, we present T-Reqs, an approach to rely on textual requirements based on a markdown format and to manage those in git. We demonstrate that this approach addresses critical challenges in large-scale agile system development. T-Reqs is proven in practice, especially in an environment that can rely mainly on textual requirements. However, models do exist and those

---

2Partially supported by Software Center: [https://www.software-center.se](https://www.software-center.se)
cannot be as easily managed with the proposed solution, asking for future research. Given the specific environment from which we draw our experience, we cannot reason about the value of T-Reqs in other industrial contexts. Regardless, we believe that we can facilitate a discussion of changing needs towards tooling in agile development.
A Simplified Guide to the use of T-Reqs
Rashidah Kasauli and Eric Knauss

Technical report, arXiv, 2020

Abstract

This document elaborates on the usage and concept behind the Text-based REQuirements System (T-Reqs) tool as it was published at RE’18. Basing on the main message of our previous research on RE in large-scale agile systems development – bridging the gap between system-level requirements and agile teams – we elaborate on two of the challenges that relate. These are: 1) updating requirements and 2) tooling not fit for purpose. Through illustrating how the tool helps to update the requirements, we thus demonstrate that this customisable tool is a good drive towards defining agile tools.
7.5 Towards Agile Tooling for RE

Requirements engineering (RE) is crucial to support agile development of large systems with long lifetimes. However, research shows that companies struggle to get a clear process of dealing with requirements in large-scale agile environments. Specifically, there is a challenge of bridging the gap between the agile development teams and the not so agile system-level requirements. The agile teams must be aware of requirements at system level and be able to propose updates to those requirements. Currently, tools used at system level differ from those used by the agile teams. Although there is a plethora of tools for software engineering, tools for managing RE in a scaled agile environment are still lacking.

Text-based Requirements system (T-Reqs) uses the Git version control system to support simultaneous work of multiple users on the same artefact. It is currently in industrial use and based on its success, we explored and continue to explore whether providing its key concepts and infrastructure as open source is valuable. Regardless, we hope our experience helps to trigger a dialogue on how the changing landscape of RE in relation to agile system and software development affects needs for tool support.

7.6 Overview

T-Reqs is a requirements management tool based on Git. It gives cross-functional teams an environment that they can relate to (as they are often used to work with Git) and that allows them to work with requirements in an iterative, incremental manner within a systems development environment. It allows developers to work concurrently and also propose changes to the requirement within the same environment they are used to. T-Reqs is an extension to the existing Git infrastructure that enables storage of requirements, additional test case information and traces between artifacts in the product code repository. Requirement data is stored in text files, but test case data can be stored in source code files as well. It utilizes Git to deliver, merge and review requirements and test cases just like source code. The overview of T-Reqs is as shown in Figure 7.1 and the process flow detailed in Figure 7.2.
T-Reqs provides templates for creation of the required documents, e.g. user stories, tests, system requirements. These templates require IDs for the respective documents, e.g. requirements’ ID for requirements documents. Along with these IDs, the IDs of any documents connected to the current document should also be provided. While defining the respective IDs, these IDs and any of their dependencies should appear on the first line of the document, as that is where T-Reqs searches while comparing documents. For example, `testcase id=TC1 TestCaseName story=US1 req=REQ1` could be the first line in a test case file. Figure 7.3 illustrates the use of IDs in T-Reqs documents.

### 7.7 Example elaboration

In this section, we demonstrate how T-Reqs allows to tackle some of the key challenges of large-scale agile system development. Specifically we elaborate on the challenge of updating requirements and on the challenge of tooling not fit for purpose.

#### 7.7.1 Updating requirements

We take the case of system requirements that typically span many teams. In most cases, teams are working with already existing requirements, and in some rare cases a team may create a new requirement. For the existing requirements, a “general” system manager, who is not connected to any team, writes the requirements beforehand. When it is time for a team to implement that requirement, there is handover from the system manager to the team. During that handover, the requirements are refined so that they are clear and testable. During the refinement, requirements are discussed among team members, through T-Reqs. The system manager then approves the modified document. The team then starts to derive user stories for development.
Figure 7.3: Linking tests and requirements to tests via T-Reqs

A developer can work with the user story and push the code to the system which is then verified by the other team members for consistency with the requirement. In case the user story is unclear, a member can also propose changes which have to be approved by the peers before merging it to the repository. For instance, we could have a user story defined as “As a member of a cross-functional team, I want to be aware of requirements changes that affect my team so that we can pro-actively address dependencies.” To address the requirements implied by this user story, T-Reqs relies on Git to group changes (e.g. to source code and tests) into commits that can be pushed to the main branch. The other team members (and members of several other teams) can see the proposed changes and either agree or decline before merging them to the main branch. Then T-Reqs can be run on to give the list of the requirements and respective tests with the links between them. Any commit that is not traced yet will be evident as well as requirements that have not been tested as illustrated in Fig 7.6.

7.7.2 Tooling not fit for purpose

The T-Reqs tool aims to integrate well with the workflow of an agile team. For instance, T-Reqs handles updates of requirements in a way that is integrated in the typical tooling of agile teams. Through pulling updates from the main branch, new system requirements updates are merged into the view of teams. If a team has to update requirements, these changes can be pushed towards the main branch and then be reviewed by system level roles and other teams. In this way, T-Reqs is a tool that supports fast requirement changes without affecting the speed and pace of work. While working within the same tool, the problem of losing information with change of tools is considerably reduced.

To that effect, T-Reqs, as an agile tooling solution offers some benefits. One major benefit of T-Reqs is that all information that pertains to a certain test case can be stored in the same file. Also, since everything is stored in text format, writing scripts to automate updates is easy.

7.8 Demonstration

Since TReqs targets large-scale agile system development, it is not easy to
Figure 7.4: Defining functional requirement template

showcase its key features. Here, we choose a simplistic and small example that allows us to discuss how TReqs integrates with an agile development approach. We take a classic exercise from eXtreme Programming, the roman number kata\(^3\). This exercise shows how several practices of eXtreme programming interact in a meaningful way. The task is to convert between roman and arabic numbers. Developers would first write a test, then code until the test is passed, then refactor, before extending the test. Thus, this example is inherently iterative.

As demonstration to usage of the tool, we acted as different roles where each of us was assumed to be from a different team. We wrote a high-level script that demonstrates how T-Reqs is addressing RE challenges of scaled agile. We agreed and defined a high-level requirement to convert roman numbers to arabic numerals. The key issues we had to take care of with T-Reqs: 1) keep code, tests and requirements in sync and traceable, 2) make other teams and system/product level roles aware of changes that concern them and 3) quality assure requirements. So we set roles Y, E, and R.

Teams led by Y, R, and E meet to discuss a new product that should be developed in an agile way. They align on a brief vision and set a high-level goal for the first sprint (see Fig. 7.4).

Based on this vision, they create a user story (US1) and while they are working, a new user story (US2) is brought up which the team(s) discuss until they realise they need to update the high-level requirement in order to make consistent user stories. This is illustrated in Figure 7.5. On running T-Reqs on the above problem, a list of the available user stories and their links to the test and requirements was revealed as presented in Figure 7.6.

T-Reqs requires that files are saved in a particular format. As it is now, all test case files should begin with ‘TC’, user story file begin with ‘US’ and system requirement files should end with ‘sys.reqt’ and all of them should be markdown files. The test files however are also checked if they are source code files. In this version of T-Reqs, we also search ‘TC..*.py’ files. This can however be customised to the company or project preferred format. Other parameters

\(^3\)http://codingdojo.org/kata/RomanNumerals/
Break down requirements and keep them consistent with implementation.

Figure 7.5: System requirement update issue

Figure 7.6: T-Reqs report file sample
7.9 Limitations and future work

T-Reqs has been recently extended to include model generation to help in the visualisation of traces and links between users and requirements. This extension also includes automatic generation of IDs for the test cases, requirements, user stories and any other documents developers could be required to use. In its initial form, T-Reqs has manual creation of IDs which could be cumbersome and hard to monitor. The automatic ID extension uses an external server (AWS) which requires creating an account. We see that as extra effort for the developers and are working towards a solution that makes ID creation simpler. Given the specific environment from which we draw our experience, we still cannot reason about the value of T-Reqs in other industrial contexts. Regardless, we believe that we can facilitate a discussion of changing needs towards tooling in agile development.
Bibliography


