The Journal of Systems & Software 172 (2021) 110851

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Requirements engineering challenges and practices in large-scale agile system development

Rashidah Kasauli [a,*], Eric Knauss [a], Jennifer Horkoff [a], Grischa Liebel [b], Francisco Gomes de Oliveira Neto [a]

[a] Department of Computer Science and Engineering, Chalmers | University of Gothenburg, Gothenburg, Sweden
[b] School of Computer Science, Reykjavik University, Menntavegur 1, 102 Reykjavik, Iceland

ARTICLE INFO

Article history:
Received 26 February 2020
Received in revised form 20 August 2020
Accepted 21 October 2020
Available online 29 October 2020

Keywords:
Requirements engineering
Large-scale agile
Systems engineering

ABSTRACT

Context: 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.
Objective: This paper presents a multiple case study with seven large-scale systems companies, reporting their challenges, together with best practices from industry. We also analyze literature about two popular large-scale agile frameworks, SAFe® and LeSS, to derive potential solutions for the challenges.
Methods: Our results are based on 20 qualitative interviews, five focus groups, and eight cross-company workshops which we used to both collect and validate our results.
Results: We found 24 challenges which we grouped in six themes, then mapped to solutions from SAFe®, LeSS, and our companies, when available.
Conclusion: In this way, we contribute a comprehensive overview of RE challenges in relation to large-scale agile system development, evaluate the degree to which they have been addressed, and outline research gaps. We expect these results to be useful for practitioners who are responsible for designing processes, methods, or tools for large scale agile development as well as guidance for researchers.

1. Introduction

Despite wide criticism, agile approaches have significantly contributed to the way software is developed (Meyer, 2014). While initially focused on small teams (Meyer, 2014; Kahkonen, 2004; Beck, 2000; Paasivaara and Lassenius, 2016), success stories have led to their application at large-scale (Dikert et al., 2016; Lagerberg et al., 2013; Salo and Abrahamsson, 2008) and in system development (i.e., large, complex systems which mix software and hardware) (Eklund et al., 2014; Berger and Eklund, 2015; Lagerberg et al., 2013), an environment that is characterized by long lead times (Berger and Eklund, 2015) and stable, sequential engineering practices (Pernstål et al., 2012). 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 (Kasauli et al., 2017b), and managing requirements (Savolainen et al., 2010). Because of these and other challenges, companies struggle to implement efficient RE in a large-scale agile context (Laanti et al., 2011; Wiklund et al., 2013; Chow and Cao, 2008).

Existing work looking at RE-related challenges arising from agile methods, i.e. agile RE (e.g., Ramesh et al., 2010; Heikkilä et al., 2017; Bjarnason et al., 2011), mostly focus on proposing new approaches, practices, and artifacts (Heikkilä et al., 2015). 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 (Heikkilä et al., 2017; Heikkilä et al., 2015; Inayat et al., 2015). This gap is a major obstacle when transitioning to agile system development at scale, considering the extraordinary demands on long-term maintenance, synchronization 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.

Through a multiple case study of seven large-scale system development cases, based on five focus groups, eight cross-company workshops and 20 semi-structured interviews, as well as a review of state-of-the-art large-scale agile frameworks, this paper makes

* Corresponding author.
E-mail addresses: rashida@chalmers.se (R. Kasauli), eric.knauss@cse.gu.se (E. Knauss).

https://doi.org/10.1016/j.jss.2020.110851

three contributions from an RE perspective. First, we present a report of industrial RE challenges related to applying agile development in large-scale systems. The identified challenges fall roughly into the areas of building and maintaining shared knowledge, representing that knowledge, as well as integrating it into the process and organization.

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 revises and extends our previous work (Kasauli et al., 2017b) by two more six-month iterations with three additional companies.

This paper is organized as follows. In Section 2, we discuss the background of large-scale agile development and the related works to our study. Section 3 presents the research questions as well as the methodology we used including data collection, analysis and validity threats. We provide study context and case company agile pervasiveness in Section 4. In Section 5, we report the results to our research questions. We discuss our results in Section 6, before concluding our article in Section 7.

2. Related work

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 six development teams (Dikert et al., 2016) and is aligned with agile principles (Meyer, 2014). To that effect, this section discusses the background of RE and agile development in large-scale systems engineering companies. We start by giving the background of agile development in large-scale systems engineering. We then discuss the background of RE and agile development while also identifying related works in that context. A summary of the related work with respect to our research then concludes the section.

2.1. Large-scale Agile

Agile methods like Scrum and XP are being adopted in large-scale system development companies (Salo and Abrahamsson, 2008), even though they were originally intended for use on a small scale (Kahkonen, 2004; Beck, 2000; Paasivaara and Lassenius, 2016). 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 (Salo and Abrahamsson, 2008) report successful adoption of these methods and appreciation among practitioners. Lindvall et al. (2004) 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. (2013) 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 (2018) analyze 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. (2016) 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. (2018) 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. (2014), 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 (2015) 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 (Ebert and Paasivaara, 2017) such as Scaled Agile Framework (SAFe®) (Leffingwell, 2010; Knaster and Leffingwell, 2017) and Large-Scale Scrum (LeSS) (Larman and Vodde, 2016). 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.

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. (2015) 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 (Heikkilä et al., 2017). 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. (2011) 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. (2010) identify risks with the use of agile RE. These are, e.g., the neglection 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 (Inayat et al., 2015), 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 (Paetsch et al., 2003) 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 (Meyer, 2014), however not based on empirical data.

2.3. Summary of related work

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 (Heikkilä et al., 2015; Inayat et al., 2015), or at presenting the current state of practice at single companies (Heikkilä et al., 2017) and without explicitly targeting system development (Bjarnason et al., 2011). 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. The study extends our preliminary work presented in Kasauli et al. (2017b) as follows:

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

3. Research methodology

We have conducted our multiple-case study (Runeson et al., 2012) in two rounds with several points of elicitation and validation in each round. Overall, the elicitation took place over a period of two years, with data analysis and writing continuing for another year. In Round 1, we investigated four companies with a series of focus groups and interviews, the results of which have been summarized in Kasauli et al. (2017b). 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 two research questions:

RQ1: *Which requirements-related challenges exist in large-scale agile system development?* Given the scope of agile development at the different case companies, we categorize and describe the challenges provided by our case companies.

RQ2: *Which approaches have been proposed in popular literature and which approaches are used by practitioners to address the challenges identified in RQ1?* Using the results from RQ1 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 (Larman and Vodde, 2016) and SAFe® (Knaster and Leffingwell, 2017).

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 technology and engineering company (Technology 2), one manufacturing company (Manufacturing) and one processing company (Processing). Both Manufacturing 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-intense systems. We elaborate on the specific cases in Section 4.

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. The elicitation and validation for this study was conducted in two rounds as elaborated in Sections 3.2.1 and 3.2.2.

3.2.1. Round one elicitation and validation

Fig. 1 gives an overview of our research design for the first round of elicitation and validation, as presented in Kasauli et al. (2017b), and Table 1 presents a summary of the individual elicitation events.

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.

Table 1
Data sources first round (reported in Kasauli et al., 2017b).

Type	Company	Role(s)	Label
Focus group	Telecom	2xTest Architect, System Manager	FG-1
Focus group	Automotive 1	Process Manager, Specialist Platform Software	FG-2
Focus group	Telecom	2xTest Architect, System Manager	FG-3
Focus group	Automotive 2	System Responsible, 2x Function Owner, System Quality Engineer	FG-4
Focus group	Technology 1	RE Change Agent, Chief Engineer	FG-5
WS	Automotive 1	Verification Manager, Specialist Platform Software	XComp 1
	Telecom	Test Architect, System Manager	
WS	Automotive 1	Verification Manager, Specialist Platform Software	XComp 2
	Telecom	Test Architect, System Manager	
	Automotive 2	Test Architect, System Manager	
	Technology 1	Chief Engineer Software	
Int	Telecom	Test Architect (TA), System Manager (SysM) x2, Developer (T-Dev), Scrum Master (ScrM), Area Product Owner (APO), Operational Product Owner (OPO),	T-*
Int	Automotive 1	Safety Technology Specialist (TS),	A1-TS
Int	Automotive 2	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)	A2-*
Int	Technology 1	Requirements responsible	Tec-SRR

Table 2
Data sources second round.

Type	Company	Role(s)	Label
WS	All companies invited	Many roles, see below	XComp PV
WS	Telecom	Verification Manager, System Architect	TelWS
WS	Technology 1	Project Manager, Chief Engineer, Requirements Manager, Technical Integrator (2x), Product Manager	TechWS
WS	Automotive 2	Project Manager (Process/Methods/Tools) 3x (OD, SW-Dev, Sys-Dev), Technical Expert	Auto1WS
WS	Manufacturing	Project Manager x2, Chief Engineer, Electronic Developer, Technical Integrator x2	ManWS
WS	Processing	System Engineer x3, Technology Specialist x2, Project Manager, Requirement Manager	ProcWS
WS	Telecom, Technology 1, Technology 2, Automotive 1, Automotive 2, Processing	Toolchain and Processes, Requirements Expert, Process/Method/Tools(SysEng) x4, Process/Methods/Tools (SW) SW Reqt Eng. x3, Architect x2, System Manager	XComp 3
WS	Telecom, Technology 1, Technology 2, Automotive 2	System Engineer x2, Agile Team Lead, Product Owner x2, Agile Expert x2, Requirements Engineer x3	XComp 4

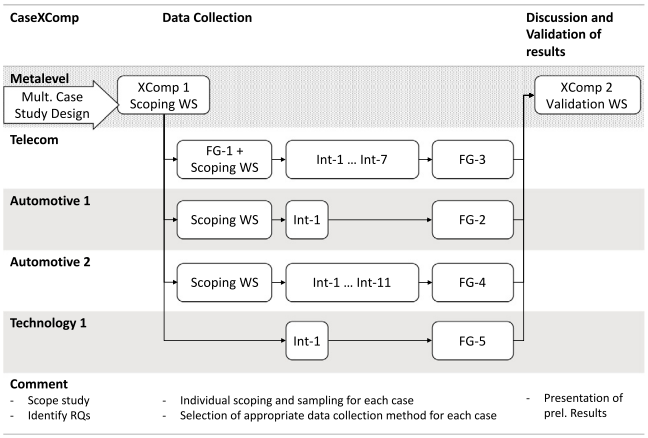

Fig. 1. Overview of multiple case study research design — Round 1.

Upon selecting the appropriate case, we started data collection through the use of interviews and focus groups. 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 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.

[1] http://www.cse.chalmers.se/~knauss/2020-AgileREChallenges/KGLKK_re_agile.pdf.

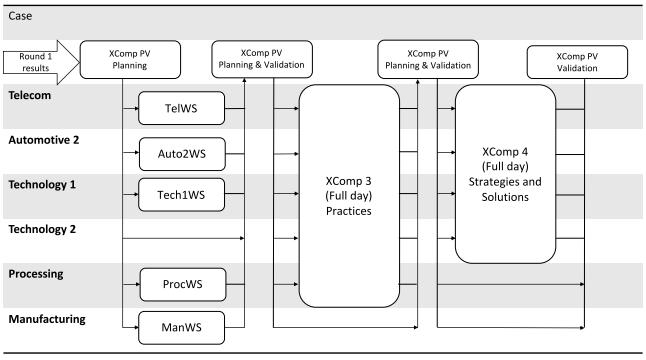

Fig. 2. Overview of multiple case study research design — Round 2.

3.2.2. Round two elicitation and validation

In the second round of data collection, summarized in Fig. 2, with elicitation events listed in Table 2, we relied mainly on workshops to expand and validate our data. We did Round 2 over a period of one and a half years divided into three six-month cycles (each column in Fig. 2). This round includes regular cross-company workshops, labeled XComp PV (Planning and Validation), that served as checkpoints to validate our findings with our industry partners and plan together the upcoming cycles. These XComp PV workshops gave us an opportunity to continually present our findings, receiving feedback. Round 1 (Fig. 1) covers six months and Round 2 (Fig. 2) covers one and a half years, making a total of two years composed of four 6-month cycles for the entire investigation.

We started Round 2 with a cross-company workshop (XComp PV) at which we presented and received feedback on our previous findings, making plans for further investigation with each company. We then conducted a number of individual workshops with each company, following a standard workshop instrument.[2] 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. Using this instrument as a guide, we conducted five company workshops (TelWS, TechWS, Auto1WS, ManWS, ProcWS) confirming, expanding and collecting challenges and solutions. Each workshop lasted three hours and was hosted by the case company.

At this stage, with a tentative list of challenges and potential solutions from industry, we sought for solutions from available literature. 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 (Larman and Vodde, 2016) and SAFe® (Knaster and Leffingwell, 2017). 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 RQ2.

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. At these workshops, 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[3] focusing on selected issues. Depending on availability, not all case companies could send representatives to each workshop, but each workshop had at least four companies represented (see Fig. 2).

3.3. Data verification

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 workshops.

3.4. Data analysis

For data analysis, we relied on a thematic coding approach (Gibbs, 2008). 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 Kasauli et al. (2017b) 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, we processed the material collected from company workshops in the same manner, i.e., with two researchers sorting and updating findings to create the updated list of issues and solutions presented in Section 5. In order to validate the clusters in each round, we discussed the outcome of our analysis in a reporting workshop (XComp 2 Validation WS for Round 1 and XComp PV for Round 2) 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 5.3.3. 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 Kasauli et al. (2017b), 'C2.c Avoid Re-specifying, Encourage Re-use', and is integrated with the other explanatory text in Section 5.2.3.

Overall, comparing Round 1 to Round 2, we re-worded and re-organized several challenges, as well as adding many new categories and sub-categories. Of the six challenge categories with 24 sub-challenges presented in this work, two of the challenges categories and roughly 11 of the sub-categories appeared already in the Round 1 results (details can be found in Kasauli et al. (2017b)). As such, roughly half of the challenges arose in Round 2, and are new to this work. Overall, due to the extensive changes, categories and challenges from Round 1 appear in a more elaborate context and are adjusted accordingly, which makes a clear mapping between Round 1 and Round 2 results difficult.

[2] http://www.cse.chalmers.se/~knauss/2020-AgileREChallenges/SWC27-Sprint12-Interview-Instrument.pdf.

[3] The world cafe method allows to effectively host large group discussions http://www.theworldcafe.com/key-concepts-resources/world-cafe-method/.

Once the challenges were established, we repeated the analysis process going through material collected from company workshops and the LeSS (Larman and Vodde, 2016) and SAFe® (Knaster and Leffingwell, 2017) 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 presents both the challenges, updated from Kasauli et al. (2017b), 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.

3.5. 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 regularly 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 designed and improved the interview guides in multiple iterations and with correspondence from the company contacts that, as mentioned, are knowledgeable with agile transformation. 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. While it is important to maintain a chain of evidence from the data to findings, we did not attempt to connect all found challenges to specific cases. Since many data points came from cross-company workshops or focus groups with many participating companies, it was not always possible to clearly decide which company had this challenge. We could have tried to establish these connections, e.g., by following up with a survey. However, we refrained from doing so to avoid confirmation bias or bandwagon effect, i.e., that company representatives would agree to challenges simply because they sound likely and because others experience them as well.

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 analyzed 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 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.

4. Study context: Pervasiveness of Agile development

Before answering the RQs as outlined in Section 3, we need to get an overview of how the case companies work with agile in a large-scale, setting the context of the study. For this, we analyzed our collected data and confirmed our summaries with the company representatives. We found it challenging to characterize 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. Fig. 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® Leffingwell et al., 2014). 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

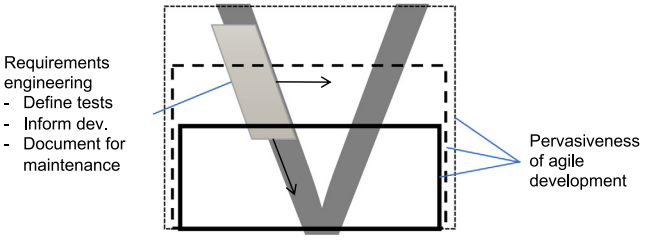

Fig. 3. Different scopes of agile development within system development and a typical V model. (Light gray box denotes RE, arrows indicate how requirements are used for informing developers about what to implement, testers about what to test, and for documenting the system for maintenance. Three black boxes show the different agile scopes discussed in this case study.).

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 Fig. 3 this is shown by the largest outer box encompassing the entire V model, 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 Fig. 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 Eklund et al., 2014) in FG-4. In Fig. 3, we describe this as the smallest box, not to refer to overall development speed, but to the fact that handover 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 Fig. 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. 3.

Processing company. The processing company, Processing , 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 Fig. 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 (Ståhl and Bosch, 2014). Others are currently moving in that direction. Our research aims for common themes, regardless of the pervasiveness of agile adoption 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. Challenges and potential solutions (RQ1 and RQ2)

With respect to RQ1, we see 24 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*. For each challenge, we also discuss potential solutions from literature and practice to answer RQ2.

In this section, we present the challenges along with solutions candidates from SAFe® and LESS, and, when available, solutions suggested by our participating companies. It is our goal to provide a comprehensive overview. 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. Fig. 4 provides an overview of the categories and challenges, and, in addition, we summarize challenges,

(C1) Build and maintain shared understanding of customer value
a) Bridge gap to customer
   i. *Make team understand customer value*
   ii. *Unable to express value in user stories*
   iii. *Feedback and clarification*
b) Building long-lasting customer knowledge

(C2) Support Change and Evolution
a) Managing experimental requirements
b) Synchronization of development
c) Avoid re-specifying, encourage re-use
d) Updating requirements

(C3) Build and maintain shared understanding about system
a) Documentation to complement tests and stories
b) System vs. component thinking
c) Creating and maintaining traces
d) Learning and long-term knowledge
e) Backward compatibility

(C4) Representation of reqts knowledge
a) Manage levels vs. decomposition
b) Quality reqts as thresholds
c) Tooling not fit for purpose
d) Accommodate different representations
e) Consistent reqts quality

(C5) Process aspects
a) Prioritization of distributed functionality
b) Manage completeness
c) Consistent requirements processes
d) Quality vs. time-to-market

(C6) Organizational aspects
a) Bridge plan-driven and agile
b) Plan V&V based on reqts
c) Time for invention and planning
d) Impact on infrastructure

Fig. 4. Challenging Areas of RE for Large-Scale Agile System Development.
Source: Updated from Kasauli et al. (2017b).

Table 3
Summary of results for challenge area 1: Build and maintain shared understanding of customer value.

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

potential solutions, and research gaps for each category in six summary tables, Tables 3 to 8. Readers can either follow our report sequentially, or use the figure and the tables as a starting point to directly jump to a challenge of interest.

5.1. Build and maintain shared understanding of customer value

5.1.1. 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

Table 4
Summary of results for challenge area 2: Support change and evolution.

ID	Challenge	Proposed practices from case companies	Proposed practices from SAFe	Proposed practices from LeSS	Research gap in large-scale agile
2.a	Managing experimental requirements	→Introduce Learning and increment planning Sprints; →T-Reqs: Use git branching and merging.	→Combine enabler stories, architecture, and exploration; →Set-based design; →(Variable) solution intent.	→Use backlog	→Design and evaluate an approach to manage experimental requirements.
2.b	Synchronization of development	→Differing levels of agile pervasiveness	→Biweekly ART sync; →F2f PI planning; →Enabler stories (Exploration); →Tribes, chapters, guilds; →Unity hour	→Continuous improvement; →Retrospectives (Team + overall).	→Design and evaluate an approach to synchronize development based on promising ideas in literature.
2.c	Avoid re-specifying, encourage re-use	→Product-line engineering; →Move from project to product focus.	→Product-focused; →Set based design.	→Product-focused; →Avoid duplicate product functionality; →Avoid narrow product definition.	→Strategies and guidance for systematic reuse and agile product-line engineering at scale.
2.d	Updating requirements	→T-Reqs: Reviews supported by git and gerrit.	→Guilds and chapters.	→Requirement areas will change, will get less important, have a lifecycle, be retired.	→Provide concrete advice and tools for updating requirements and to establish awareness of the current state.

Table 5
Summary of results for challenge area 3: Build and maintain shared understanding about system.

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

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 Creating and maintaining traces*. 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

Table 6
Summary of results for Challenge Area 4: Representation of requirements knowledge.

ID	Challenge	Proposed practices from case companies	Proposed practices from SAFe	Proposed practices from LeSS	Research gap in large-scale agile
4.a	Manage levels vs. decomposition	→ 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.	→Clear hierarchy, from enterprise level to team; →Requirements information model (epic-capability-feature-story): transport stakeholder view to components.	→Various splitting and refinement guides with depth 3 limit.	→Strategies and guidance for requirements decomposition, including how to manage customer and system requirements as well as on how to inter-relate them.
4.b	Quality requirements as thresholds	→Thresholds to negotiate prizes; →Trade-offs brought up by team and peer-reviewed by teams and system managers.	→NFR are constraints on program level, constraining (a) the system and the product backlog or (b) a feature and the team backlog.	–	→Strategies and guidance on managing and evolving quality requirements.
4.c	Tooling not fit for purpose	→PO updates requirements; →Team updates requirements.	–	→No software tools for sprint backlog; →Tools for large product backlogs (boards, pictures, wikis, spreadsheets).	→Tools specifically designed for large-scale scale agile practices, including reqts access control.
4.d	Accommodate different representations	–	→Teams can have individual user stories flavor; →Emphasize team independence; →Responsibility for Ways of Working, book clubs, guilds.	–	→Strategies and guidance to balance independence of teams and system level consistency.
4.e	Consistent requirements quality	→Operationalization from experience; →Peer-reviews by teams and system manager.	→Responsibility for Ways of Working, book clubs, guilds; →Community of practice to align on what is needed.	–	→Ways to share experiences on quality; →Empirical evaluation of suggested methods in practice.

Table 7
Summary of results for Challenge Area 5: Process aspects.

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

Table 8
Summary of results for Challenge Area 6: Organizational aspects.

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

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® (Knaster and Leffingwell, 2017) and LeSS (Larman and Vodde, 2016), 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 (Knaster and Leffingwell, 2017). Similarly, LeSS recommends organizations to be customer-centric (Larman and Vodde, 2016). 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 (Knaster and Leffingwell, 2017; Larman and Vodde, 2016). In addition, SAFe® suggests that teams should cover all necessary roles (Knaster and Leffingwell, 2017), 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 (Larman and Vodde, 2016), 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 (Larman and Vodde, 2016), but does not share concrete advice or tools for product owners to navigate the challenges we bring up.

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 (Leffingwell, 2010).

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.

5.2. Support change and evolution

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 un-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 (Knauss et al., 2018). 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 (Knaster and Leffingwell, 2017, p. 108). Further, set-based design allows to some extent to reason about different alternatives during the development flow (Knaster and Leffingwell, 2017, p. 178). Input from exploration, research, and pre-development can also be managed as *(variable) solution intents* (Knaster and Leffingwell, 2017, p. 186ff).

LeSS, in contrast, appears to suggest that this complex topic can be handled using a backlog (Larman and Vodde, 2016).

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.

5.2.2. 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 trade-offs. SAFe® suggests to provide such synchronization through bi-weekly synchronization of agile release trains (ART, a set of agile teams that work together towards a shared release schedule) (Knaster and Leffingwell, 2017). This is further supported through enabler stories for exploration (Knaster and Leffingwell, 2017, 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) (Campbell-Pretty, 2016, 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 (Campbell-Pretty, 2016, 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 (Larman and Vodde, 2016, p. 69).

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 (Knaster and Leffingwell, 2017, 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 (Larman and Vodde, 2016, p. 159).

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

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 (Larman and Vodde, 2016, 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 (Campbell-Pretty, 2016, 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 (Knauss et al., 2018). 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 C4.c *Tooling not fit for purpose*

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.

5.3.1. 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 (Meyer, 2014) 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 solution. With respect to complementing user stories and tests, the focus groups yielded suggestions relating to two areas: modeling customer-value (see C1.a *Bridge gap to customer* and C1.b *Building lasting customer knowledge*) and modeling (distributed) system behavior. With respect to the latter, the reasoning is that teams could use models to explicitly describe the intended behavior 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 behaviors to account for their importance. There have also been further suggestions in the large-scale agile literature to use models to analyze requirements (Leffingwell, 2010, 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.

5.3.2. 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 sees a clear 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 (Knaster and Leffingwell, 2017), which, when combined with awareness on how each part fits in the complete picture, could discourage localized thinking. SAFe® does promote system thinking (Knaster and Leffingwell, 2017, pg. 70f), (Larman and Vodde, 2016, 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 (Leffingwell, 2010), 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 (Campbell-Pretty, 2016, pg. 78,83).

Similarly, also LeSS emphasizes the importance of whole product focus (Larman and Vodde, 2016, 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.

5.3.3. 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 (Knaster and Leffingwell, 2017, pg. 184ff). LeSS suggests to link to wiki pages for additional information (Larman and Vodde, 2016, pg. 33), as well as to suggest to link backlog items to ancestors for maximum three levels (Larman and Vodde, 2016, pg. 204,222), but does not offer rich details on how and by whom such links are maintained.

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 (Knaster and Leffingwell, 2017). 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 (Leffingwell, 2010). SAFe® recommends supporting communities of practice, helping to share information in particular areas (Knaster and Leffingwell, 2017). The creation of chapters or guilds has also been proposed (Campbell-Pretty, 2016) 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 topic and explore feasibility (Knaster and Leffingwell, 2017, pg. 108). LeSS promotes a similar practice by discussing reflection and encouraging improvement experiments (Larman and Vodde, 2016, p. 7,20). As a form of learning, LeSS also recommends specification by example, using concrete examples instead of more abstract user stories (Larman and Vodde, 2016, p.3,254).

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 …its 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.

5.4. 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.

5.4.1. 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 (Knaster and Leffingwell, 2017, p. 70), complemented by a hierarchical view of requirements in four levels, from epics to capabilities to features to user stories that corresponds to this clear hierarchy (Knaster and Leffingwell, 2017, 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 (Leffingwell, 2010). 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 (Larman and Vodde, 2016, p. 30). LeSS advises for traceability of certain items (e.g., product backlog items have requirement area attributes which trace to their associated area) (Larman and Vodde, 2016, p. 216). However, they advise splitting requirements only to three levels (Larman and Vodde, 2016, 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.

5.4.2. 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 emphasized 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) (Leffingwell, 2010, 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.

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 endeavors 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 advices against using software tools for sprint backlogs (Larman and Vodde, 2016, p. 18, 281), but discusses tooling for large product backlogs such as boards, wikis, pictures and spreadsheets (Larman and Vodde, 2016, 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.

5.4.4. 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 5.4.3) 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 (Knaster and Leffingwell, 2017), 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 (Campbell-Pretty, 2016). We find no potential solutions for different representations of requirements in LeSS.

[4] For our purposes we treat NFRs and quality requirements as the same, a more detailed debate on this is out of our scope.

5.4.5. 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 (Campbell-Pretty, 2016), 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 (Knaster and Leffingwell, 2017, 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. Process aspects

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 (Knaster and Leffingwell, 2017, p.65,104,212). It also advocates the combination of team backlog, business values and an "interaction backlog" (Knaster and Leffingwell, 2017, p.109,127,137) and sequencing tasks based on the cost of delay (Knaster and Leffingwell, 2017, p. 175). However, it is 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 (Larman and Vodde, 2016).

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.

5.5.2. 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 (Knaster and Leffingwell, 2017, p. 77). Similarly, LeSS recommends to "take a bite", analyzing and implementing small parts of the problem, forgoing completeness in requirements (Larman and Vodde, 2016, 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 is not clear how complete or detailed requirements for an increment should be.

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 were 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 (Knaster and Leffingwell, 2017; Leffingwell, 2010). 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 (Campbell-Pretty, 2016). 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.

5.5.4. 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 (Knaster and Leffingwell, 2017, p.23, 140), and provides guidance for reducing time-to-market through value stream mapping (Knaster and Leffingwell, 2017, 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.

5.6. Organizational aspects

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 pro-active 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 *C3.c 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. TReqs as a tool solution was again mentioned as potential enabler (Knauss et al., 2018). 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 (Knaster and Leffingwell, 2017; Leffingwell, 2010). 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.

5.6.2. 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 fulfill 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 (Knaster and Leffingwell, 2017, p. 187). Adopting a more cross-functional organization, including testers or system engineers in the agile teams, would also help to alleviate these issues (Knaster and Leffingwell, 2017, 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.

5.6.3. C6.c: Time for invention and planning

> Research activities and exploration are hard to fit into development sprints but offer fundamental information towards requirements.

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 handover 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 behaviors (Knaster and Leffingwell, 2017, p. 20). SAFe® also describes both enabler stories, stories that explicitly support exploration (Knaster and Leffingwell, 2017, p. 108) and iterations dedicated to innovation and planning (Knaster and Leffingwell, 2017, 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.

5.6.4. 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 (Knaster and Leffingwell, 2017, p. 97). We do not find a potential solution in LeSS.

This challenge shows that independent of the pervasiveness in Fig. 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.

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.

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 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 (Evbota et al., 2016) as well as agile practices such as continuous delivery (Kasauli et al., 2017a) 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 (Heikkilä et al., 2015; Ramesh et al., 2010; Inayat et al., 2015), but it seems that the notion of value itself is problematic and a shared language for discussing value is needed (Kasauli et al., 2017a; Khurum et al., 2013; Horkoff et al., 2018) as well as approaches to systematically enable, build, and assess shared understanding (Batsaikhan and Lin, 2018). 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.

6.2. Support change and evolution

As summarized in Table 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 (Cockburn, 2006).

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. 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) (Sommerville, 2015). 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 (Lauesen, 2002, 2017), and we 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 (Heikkilä et al., 2015) and (automated) test cases are often named as an alternative (Heikkilä et al., 2015; Bjarnason et al., 2016), 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 Heikkilä et al. (2017). 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 (Ambler, 2002; Rumpe, 2004), 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 (Inayat et al., 2015) 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 (Meyer, 2014; Heikkilä et al., 2017). Similarly, we did not find any specific roles that emerge in the large-scale agile environment comparable to the roles presented in Hoda et al. (2013).

Our results suggest that continuous and agile development methods on a large scale require new concepts. Hybrid approaches (Kuhrmann et al., 2017) 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 (Wohlrab et al., 2019; Lindman et al., 2020), 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 (Wohlrab et al., 2019). We have explored this approach with respect to strategic API management and governance (Lindman et al., 2020) and we are confident that this research direction will yield useful concepts and theories to tackle this challenge area.

6.4. Representation of requirements knowledge

The fourth challenge area includes challenges that relate to the representation of requirements knowledge (see Table 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 flavors 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 (Wohlrab et al., 2020).

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 (Knauss et al., 2018). 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., Lucassen et al. (2015), 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., Inayat et al. (2015) and Bjarnason et al. (2011).

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* (Davis, 2005) 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 (Bjarnason et al., 2011; Heikkilä et al., 2015; Inayat et al., 2015). 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 and Hassanzadeh (2019), 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 (Sedano et al., 2019; Wohlrab et al., 2019) and bridging methodological gaps between different scopes in large-scale agile (Kasauli et al., 2020) may offer useful guidelines.

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 (Sedano et al., 2019; Wohlrab et al., 2019). If constructively used to establish boundary objects as means of coordination between plan-driven and agile areas of an organization, we expect a positive impact on organizational aspects with engineering requirements in scaled-agile system development (Kasauli et al., 2020). Our works on T-Reqs can be seen as a special boundary objects, where teams can communicate critical requirements changes early on and spread awareness through peer reviews (Knauss et al., 2018).

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 catalog 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 (Meyer, 2014). 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 (Inayat et al., 2015; Heikkilä et al., 2015), and first works exist to address regulations in agile (Hanssen et al., 2016; Fitzgerald et al., 2013). 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 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 (ISO, 2018). 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 (Kasauli et al., 2018; Steghöfer et al., 2019).

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. Thus, each organization must find workarounds for their particular context. Our results facilitate the search for such individual solution strategies by providing a comprehensive overview of challenges. In particular, it helps to design solutions that do not over-optimize solving one challenge at the expense of a different challenge. Further inspiration is provided by listing relevant solution candidates. Yet, more general and reusable approaches are desperately needed. Therefore, 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.

CRediT authorship contribution statement

Rashidah Kasauli: Methodology, Formal analysis, Investigation, Data curation, Writing - original draft, Writing - review & editing, Visualization. **Eric Knauss:** Conceptualization, Methodology, Validation, Formal analysis, Investigation, Writing - original draft, Writing - review & editing, Supervision, Project administration, Funding acquisition. **Jennifer Horkoff:** Conceptualization, Methodology, Validation, Formal analysis, Investigation, Data curation, Writing - original draft, Writing - review & editing. **Grischa Liebel:** Methodology, Formal analysis, Investigation, Data curation, Writing - original draft, Writing - review & editing. **Francisco Gomes de Oliveira Neto:** Methodology, Investigation, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

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 Requirements Engineering for Large-Scale Agile System Development and the Sida/BRIGHT project 317 "Building Research Capacity in Innovative Information and Communication Technologies for Development (ICT4D) for Sustainable Socio-economic Growth in Uganda (BRIGHT)" under the Sida contribution No: 51180060.

References

Ambler, S., 2002. Agile Modeling: Effective Practices for Extreme Programming and the Unified Process. John Wiley & Sons.

Batsaikhan, O., Lin, Y.-C., 2018. Building a Shared Understanding of Customer Value in a Large-Scale Agile Organization: A Case Study (master thesis). Chalmers | University of Gothenburg, Dept. of Computer Science and Engineering.

Beck, K., 2000. Extreme Programming Explained: Embrace Change. Addison Wesley Longman Inc..

Berger, C., Eklund, U., 2015. Expectations and challenges from scaling agile in mechatronics-driven companies – A comparative case study. In: Lassenius, C., Dingsøyr, T., Paasivaara, M. (Eds.), Agile Processes in Software Engineering and Extreme Programming. Springer International Publishing, Cham, pp. 15–26.

Bjarnason, E., Unterkalmsteiner, M., Borg, M., Engström, E., 2016. A multi-case study of agile requirements engineering and the use of test cases as requirements. Inf. Softw. Technol. 77, 61–79. http://dx.doi.org/10.1016/j.infsof.2016.03.008.

Bjarnason, E., Wnuk, K., Regnell, B., 2011. A case study on benefits and side-effects of agile practices in large-scale requirements engineering. In: Proceedings of the 1st Workshop on Agile Requirements Engineering. In: AREW '11, Association for Computing Machinery, New York, NY, USA, pp. 1–5. http://dx.doi.org/10.1145/2068783.2068786.

Campbell-Pretty, 2016. Tribal Unity: Getting from Teams to Tribes by Creating a One Team Culture. CreateSpace Independent Publishing Platform.

Chow, T., Cao, D.-B., 2008. A survey study of critical success factors in agile software projects. J. Syst. Softw. 81 (6), 961–971. http://dx.doi.org/10.1016/j.jss.2007.08.020, Agile Product Line Engineering.

Cockburn, A., 2006. Agile Software Development: The Cooperative Game. Pearson Education.

Davis, A.M., 2005. Just Enough Requirements Management: Where Software Development Meets Marketing. Dorset House.

Dikert, K., Paasivaara, M., Lassenius, C., 2016. Challenges and success factors for large-scale agile transformations: A systematic literature review. J. Syst. Softw. 119, 87–108. http://dx.doi.org/10.1016/j.jss.2016.06.013.

Ebert, C., Paasivaara, M., 2017. Scaling agile. IEEE Softw. 34 (6), 98–103. http://dx.doi.org/10.1109/MS.2017.4121226.
Eklund, U., Holmström Olsson, H., Strøm, N.J., 2014. Industrial challenges of scaling agile in mass-produced embedded systems. In: Dingsøyr, T., Moe, N.B., Tonelli, R., Counsell, S., Gencel, C., Petersen, K. (Eds.), Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation. Springer International Publishing, Cham, pp. 30–42.
Evbota, F., Knauss, E., Sandberg, A., 2016. Scaling up the planning game: Collaboration challenges in large-scale agile product development. In: Sharp, H., Hall, T. (Eds.), Agile Processes, in Software Engineering, and Extreme Programming. Springer International Publishing, Cham, pp. 28–38.
Fitzgerald, B., Stol, K.-J., O'Sullivan, R., O'Brien, D., 2013. Scaling agile methods to regulated environments: An industry case study. In: 35th International Conference on Software Engineering (ICSE). IEEE, pp. 863–872. http://dx.doi.org/10.1109/ICSE.2013.6606635.
Gibbs, G.R., 2008. Analysing Qualitative Data. Sage.
Hadar, E., Hassanzadeh, A., 2019. Big data analytics on cyber attack graphs for prioritizing agile security requirements. In: IEEE 27th International Requirements Engineering Conference (RE). Jeju Island, South Korea. pp. 330–339. http://dx.doi.org/10.1109/RE.2019.00042.
Hanssen, G.K., Haugset, B., Stålhane, T., Myklebust, T., Kulbrandstad, I., 2016. Quality assurance in scrum applied to safety critical software. In: Sharp, H., Hall, T. (Eds.), Agile Processes, in Software Engineering, and Extreme Programming. Springer International Publishing, Cham, pp. 92–103.
Heikkilä, V.T., Damian, D., Lassenius, C., Paasivaara, M., 2015. A mapping study on requirements engineering in agile software development. In: 41st Euromicro Conference on Software Engineering and Advanced Applications. pp. 199–207. http://dx.doi.org/10.1109/SEAA.2015.70.
Heikkilä, V.T., Paasivaara, M., Lasssenius, C., Damian, D., Engblom, C., 2017. Managing the requirements flow from strategy to release in large-scale agile development: a case study at Ericsson. Empir. Softw. Eng. 22, 2892–2936. http://dx.doi.org/10.1007/s10664-016-9491-z.
Hoda, R., Noble, J., Marshall, S., 2013. Self-organizing roles on agile software development teams. IEEE Trans. Softw. Eng. 39 (3), 422–444. http://dx.doi.org/10.1109/TSE.2012.30.
Horkoff, J., Lindman, J., Hammouda, I., Knauss, E., 2018. Experiences applying e^3 value modeling in a cross-company study. In: Trujillo, J.C., Davis, K.C., Du, X., Li, Z., Ling, T.W., Li, G., Lee, M.L. (Eds.), Conceptual Modeling. Springer International Publishing, Cham, pp. 610–625.
Inayat, I., Salim, S.S., Marczak, S., Daneva, M., Shamshirband, S., 2015. A systematic literature review on agile requirements engineering practices and challenges. Comput. Hum. Behav. 51, 915–929. http://dx.doi.org/10.1016/j.chb.2014.10.046, Computing for Human Learning, Behaviour and Collaboration in the Social and Mobile Networks Era.
ISO, 2018. ISO 26262:2018: Road Vehicles – Functional Safety. ISO, Geneva, Switzerland.
Jørgensen, M., 2018. Do agile methods work for large software projects? In: Garbajosa, J., Wang, X., Aguiar, A. (Eds.), Agile Processes in Software Engineering and Extreme Programming. Springer International Publishing, Cham, pp. 179–190.
Kahkonen, T., 2004. Agile methods for large organizations - building communities of practice. In: Agile Development Conference. pp. 2–10. http://dx.doi.org/10.1109/ADEVC.2004.4.
Kasauli, R., Knauss, E., Kanagwa, B., Nilsson, A., Calikli, G., 2018. Safety-critical systems and agile development: A mapping study. In: 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE, pp. 470–477. http://dx.doi.org/10.1109/SEAA.2018.00082.
Kasauli, R., Knauss, E., Nilsson, A., Klug, S., 2017a. Adding value every sprint: A case study on large-scale continuous requirements engineering. In: Proceedings of 3rd Workshop on Continuous Requirements Engineering. Essen, Germany. p. 10.
Kasauli, R., Liebel, G., Knauss, E., Gopakumar, S., Kanagwa, B., 2017b. Requirements engineering challenges in large-scale agile system development. In: IEEE 25th International Requirements Engineering Conference (RE). pp. 352–361. http://dx.doi.org/10.1109/RE.2017.60.
Kasauli, R., Wohlrab, R., Knauss, E., Steghöfer, J.-P., Horkoff, J., Maro, S., 2020. Charting coordination needs in large-scale agile organisations with boundary objects and methodological islands. In: Proceedings of the International Conference on Software and System Processes (ICSSP '20)'. pp. 51–60. http://dx.doi.org/10.1145/3379177.3388897.
Khurum, M., Gorschek, T., Wilson, M., 2013. The software value map—an exhaustive collection of value aspects for the development of software intensive products. J. Softw.: Evol. Process 25 (7), 711–741. http://dx.doi.org/10.1002/smr.1560.
Knaster, R., Leffingwell, D., 2017. SAFe 4.0 Distilled: Applying the Scaled Agile Framework for Lean Software and Systems Engineering. Addison-Wesley Professional.
Knauss, E., Liebel, G., Horkoff, J., Wohlrab, R., Kasauli, R., Lange, F., Gildert, P., 2018. T-Reqs: Tool support for managing requirements in large-scale agile system development. In: IEEE 26th International Requirements Engineering Conference (RE'18). Banff, Canada. pp. 502–503. Demo Track. http://dx.doi.org/10.1109/RE.2018.00073.
Kuhrmann, M., Diebold, P., Münch, J., Tell, P., Garousi, V., Felderer, M., Trektere, K., McCaffery, F., Linssen, O., Hanser, E., Prause, C., 2017. Hybrid software and system development in practice: Waterfall,scrum, and beyond. In: Proc. of Int. Conf. on Software System Process (ICSSP). pp. 30–39. http://dx.doi.org/10.1145/3084100.3084104.
Laanti, M., Salo, O., Abrahamsson, P., 2011. Agile methods rapidly replacing traditional methods at Nokia: A survey of opinions on agile transformation. Inf. Softw. Technol. 53 (3), 276–290. http://dx.doi.org/10.1016/j.infsof.2010.11.010.
Lagerberg, L., Skude, T., Emanuelsson, P., Sandahl, K., Ståhl, D., 2013. The impact of agile principles and practices on large-scale software development projects: A multiple-case study of two projects at ericsson. In: 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 348–356. http://dx.doi.org/10.1109/ESEM.2013.53.
Larman, C., Vodde, B., 2016. Large-Scale Scrum: More with LeSS. Addison-Wesley Professional.
Lauesen, S., 2002. Software Requirements. Pearson / Addison-Wesley.
Lauesen, S., 2017. Guide to Requirements SL-07: Problem-Oriented Requirements v5. The course book (Lau).
Leffingwell, D., 2010. Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise. Addison-Wesley Professional.
Leffingwell, D., et al., 2014. Scaled Agile Framework 3.0. Scaled Agile, Inc.
Lindman, J., Horkoff, J., Hammouda, I., Knauss, E., 2020. Emerging perspectives of application programming interface strategy: A framework to respond to business concerns. IEEE Softw. 37 (2), 52–59. http://dx.doi.org/10.1109/MS.2018.2875964.
Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, D., May, J., Kahkonen, T., 2004. Agile software development in large organizations. Computer 37 (12), 26–34. http://dx.doi.org/10.1109/MC.2004.231.
Lucassen, G., Dalpiaz, F., van der Werf, J.M.E.M., Brinkkemper, S., 2015. Forging high-quality user stories: Towards a discipline for agile requirements. In: IEEE 23rd International Requirements Engineering Conference (RE). pp. 126–135. http://dx.doi.org/10.1109/RE.2015.7320415.
Meyer, B., 2014. Agile! The Good, the Hype and the Ugly. Springer International Publishing, Switzerland, http://dx.doi.org/10.1007/978-3-319-05155-0.
Paasivaara, M., Lassenius, C., 2016. Challenges and success factors for large-scale agile transformations: A research proposal and a pilot study. In: Proceedings of the Scientific Workshop Proceedings of XP'16. Association for Computing Machinery, New York, NY, USA, p. 9. http://dx.doi.org/10.1145/2962695.2962704.
Paetsch, F., Eberlein, A., Maurer, F., 2003. Requirements engineering and agile software development. In: WET ICE 2003. Proceedings. 12th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. pp. 308–313. http://dx.doi.org/10.1109/ENABL.2003.1231428.
Pernstål, J., Magazinius, A., Gorschek, T., 2012. A study investigating challenges in the interface between product development and manufacturing in the development of software-intensive automotive systems. Int. J. Softw. Eng. Knowl. Eng. 22 (07), 965–1004. http://dx.doi.org/10.1142/S0218194012500271.
Ramesh, B., Cao, L., Baskerville, R., 2010. Agile requirements engineering practices and challenges: an empirical study. Inf. Syst. J. 20 (5), 449–480. http://dx.doi.org/10.1111/j.1365-2575.2007.00259.x.
Rumpe, B., 2004. Agile modeling with the UML. In: Wirsing, M., Knapp, A., Balsamo, S. (Eds.), Radical Innovations of Software and Systems Engineering in the Future. Springer, Berlin, Heidelberg, pp. 297–309.
Runeson, P., Höst, M., Rainer, A., Regnell, B., 2012. Case Study Research in Software Engineering, first ed. Wiley.
Salo, O., Abrahamsson, P., 2008. Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of extreme programming and scrum. IET Softw. 2, 58–64, (6).
Savolainen, J., Kuusela, J., Vilavaara, A., 2010. Transition to agile development - rediscovery of important requirements engineering practices. In: 18th IEEE International Requirements Engineering Conference. pp. 289–294. http://dx.doi.org/10.1109/RE.2010.41.
Sedano, T., Ralph, P., Péraire, C., 2019. The product backlog. In: IEEE/ACM 41st International Conference on Software Engineering (ICSE). Montreal, QC, Canada. pp. 200–211. http://dx.doi.org/10.1109/ICSE.2019.00036.
Sommerville, I., 2015. Software Engineering, tenth ed. Pearson.
Steghöfer, J.-P., Knauss, E., Horkoff, J., Wohlrab, R., 2019. Challenges of scaled agile for safety-critical systems. In: Franch, X., Männistö, T., Martínez-Fernández, S. (Eds.), Product-Focused Software Process Improvement. Springer International Publishing, Cham, pp. 350–366.
Ståhl, D., Bosch, J., 2014. Modeling continuous integration practice differences in industry software development. J. Syst. Softw. 87, 48–59. http://dx.doi.org/10.1016/j.jss.2013.08.032.
Uludag, Ö., Kleehaus, M., Caprano, C., Matthes, F., 2018. Identifying and structuring challenges in large-scale agile development based on a structured literature review. In: IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC). pp. 191–197. http://dx.doi.org/10.1109/EDOC.2018.00032.

Wiklund, K., Sundmark, D., Eldh, S., Lundqvist, K., 2013. Impediments in agile software development: An empirical investigation. In: Heidrich, J., Oivo, M., Jedlitschka, A., Baldassarre, M.T. (Eds.), Product-Focused Software Process Improvement. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 35–49.

Wohlrab, R., Pelliccione, P., Knauss, E., Larsson, M., 2019. Boundary objects and their use in agile systems engineering. J. Softw.: Evol. Process 31 (5), e2166. http://dx.doi.org/10.1002/smr.2166, e2166 smr.2166.

Wohlrab, R., Knauss, E., Pelliccione, P., 2020. Why and how to balance alignment and diversity of requirements engineering practices in automotive. J. Syst. Softw. 162, 110516. http://dx.doi.org/10.1016/j.jss.2019.110516.

Rashidah Kasauli is an Assistant Lecturer at Makerere University, Uganda. Her research focuses on requirements engineering, safety-critical systems development and large-scale agile development. She holds a Ph.D. from Chalmers University of Technology Gothenburg, Sweden. Contact her at rashida@chalmers.se.

Eric Knauss is an Associate Professor at Chalmers | University of Gothenburg, Sweden. His research focuses on managing requirements and related knowledge in large-scale and distributed software projects. He holds a Ph.D. from Leibniz Universität Hannover, Germany. He is member of program and organization committees of top conferences and reviewer for top journals. Contact him at eric.knauss@cse.gu.se.

Jennifer Horkoff is an Associate Professor at Chalmers | University of Gothenburg, Sweden. Her research focuses on requirements engineering, requirements modeling, quality and value modeling, model analysis and creativity. She holds a Ph.D. from the University of Toronto. Contact her at jennifer.horkoff@gu.se.

Grischa Liebel is an Assistant Professor at Reykjavik University, Iceland. Grischa does empirical, often qualitative, research with a focus on requirements engineering, model-based engineering, software engineering education, and human factors. He holds a Ph.D. from Chalmers University of Technology. Contact him at grischal@ru.is.

Francisco Gomes de Oliveira Neto is an Assistant Professor at Chalmers | the University of Gothenburg, Sweden. His research focuses on automated software testing and test optimization (e.g., generation, selection and prioritization of tests), particularly, to improve continuous integration pipelines and automated maintenance of test repositories. He holds a Ph.D. from the Federal University of Campina Grande. Contact him at francisco.gomes@cse.gu.se.