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

Rashidah Kasauli*
Eric Knauss*
rashida@chalmers.se
eric.knauss@cse.gu.se
Chalmers|University of Gothenburg
Gothenburg, Sweden

Joyce Nakatumba-Nabende
Benjamin Kanagwa
jnakatumba@cis.mak.ac.ug
bkanagwa@cis.mak.ac.ug
Makerere Univeristy
Kampala, Uganda

Abstract

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

CCS Concepts • **Software and its engineering → Software development methods**.

Keywords Agile islands, Hybrid methods, Co-existence

*Corresponding author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
EASE 2020, April 15–17, 2020, Trondheim, Norway
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7731-7/20/04…$15.00
https://doi.org/10.1145/3383219.3383223

ACM Reference Format:
Rashidah Kasauli, Eric Knauss, Joyce Nakatumba-Nabende, and Benjamin Kanagwa. 2020. Agile Islands in a Waterfall Environment: Challenges and Strategies in Automotive. In *Evaluation and Assessment in Software Engineering (EASE 2020), April 15–17, 2020, Trondheim, Norway*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3383219.3383223

1 Introduction

In order to meet market demands and increasing competition, large-scale systems engineering companies are adopting agile methods [5, 25, 31]. To leverage the full benefits of agility with respect to time-to-market and flexibility, such agile transformation must in the long run affect the overall systems engineering process and organization. While software teams may have quickly adopted agile methods and work with them, there is usually slow company-wide adoption, mostly attributed to the skepticism they have received [23]. Since agile teams have to work with the traditional structures within the companies, this ends up creating '*agile islands in a waterfall*' [14] – a coexistence of both agile and traditional development approaches.

However, without a company-wide adoption structure, different development teams adopt different agile practices [32]. Further, in the systems engineering context, software development is only one of several domains and must be synchronized with development of hardware and mechanics. With software teams contributing to the same product, it becomes challenging to deal with different speeds and quality measures from the different ways of working. While there is a growing body of literature on agile transformations [2, 4, 11], research addressing the coexistence of agile and traditional methods is only recently growing [18, 20, 28, 29, 31]. We are not aware of any empirical work that explores such coexistence from the development teams' perspective, especially in relation to the specific way teams adopt agile practice.

This study addresses that gap by first understanding and documenting the challenges and lessons learned based on a case study situated in the automotive domain. This domain is particularly interesting, since requirements traditionally play an important role in automotive systems, yet recent

trends around electrification, advanced driver assistance systems (ADAS), and autonomous drive (AD) as well as a market that moves towards continuous deployment of software functions render it necessary to change established engineering practices.

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

- ***RQ1:*** *What are the perceived challenges when combining plan-driven and agile paradigms in large-scale systems engineering?*
- ***RQ2:*** *What mitigation strategies exist for the challenges in RQ1?*

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

2 Related Work

Traditional methods like waterfall and V-model have formed the foundation for systems development for decades. These methods follow a sequential execution of processes with predefined phases, extensive requirements design and documentation [20, 31]. In traditional methods, RE includes a set of well defined preliminary phases dealing with analysis, planning, and documentation [7]. Meaning that requirements are supposed to be complete in the preliminary stage. However, faced with evolving market demands and fast changing requirements, systems development companies are adopting agile methods [14, 18, 29] as the traditional methods are not flexible and are generally slow due to their rigorous nature.

Agile methods encourage flexible and light-weight software development with short iterations [24]. Being flexible, agile methods can help where traditional methods fail [21]. In agile development, RE is an iterative and continuous process that is carried out in every step of development. Also, requirements are only partially known and evolve rapidly during development. Previous studies on agile and traditional development have compared the characteristics of agile development with those of traditional methods[6, 9, 15, 27] while describing the way different organizations are adopting to these methods. There are few studies concerning the coexistence of agile methodologies with traditional approaches [18–21, 28, 29, 31].

Theocharis *et al.*[29] studied the general process use over time to investigate how traditional and agile methods were used. They aimed to find out if there was coexistence or whether agile methods accelerated the traditional processes' extinction. This was done by applying instruments of systematic literature review process. They found indication of mixed application of traditional and agile methods and thus concluded that hybrid approaches that include traditional and agile approaches shaped today's "standard process ecosystem". The results of their exploratory study show that there is less material published on the combination of software development approaches than expected. This is supported by Kuhrmann *et al.*[18, 19] who present results from 69 study participants in a survey on hybrid software development approaches. They found that companies combine different development approaches regardless of the industry sector and size. These findings go beyond the adoption problem to having a state of working with both methods in coexistence. This confirms the coexistence of both methodologies.

Based on a Grounded Theory study involving 21 agile practitioners from two large enterprise organizations in the Netherlands, Waardenburg and van Vliet [31] presented challenges of using agile methods in traditional enterprise environments. They organized the challenges under two factors: increased landscape complexity and lack of business involvement, for which they identify successful mitigation strategies. These mitigation strategies concern the communication between the agile and traditional part of the organization, and the timing of that communication. Kuusinen *et al.*[21] investigate practitioners' mitigation strategies related to the challenge of doing Agile in a non-Agile environment. Through an on-line survey and a workshop, they provide strategies from both an organizational and change perspective. The strategies include: (a) ensuring managers understand and buy-in to Agile and (b) creating an organizational culture that fosters agility as the biggest themes of the study. Whereas both these studies give the organizational view, we provide the developers' perspective.

Theobald and Diebold [28] based on a literature search and workshops to identify existing interfaces of agile to non-agile environments for which they collect and group problems. They propose a classification matrix based on which we describe our research here to relate to the "project-team interface" and to provide deeper empirical knowledge about the related challenges and problem areas. Starting from a literature search, Kusters *et al.*[20] derive risks and problems at the interface of agile and traditional development approaches in hybrid organizations which have an impact on coordination and cooperation. They discover 28 issues which reduce to 22 after validation through a case study at a large financial institute in the Netherlands. They had six classifications of challenges including challenges relating to development and testing. We explore challenges in this classification, particularly software development.

In summary, the body of knowledge on the coexistence of agile and traditional methods is growing. However, to the best of our knowledge, there is a lack of empirical studies

that explore developers' experiences in this context. We believe this study will help inform a better approach to the coexistence dilemma.

3 Research Method

In this study, we explore development challenges caused by the interplay of agile and plan-driven methods and a lack of a company-wide strategy for agility. We rely on the case study method [26], which is considered appropriate when studying a complex social contemporary phenomenon for which deep understanding of the context is critical [33]. Specifically, we investigate the case of an automotive company and using two of its departments as units of analysis. We collected data through 18 semi-structured interviews and used thematic coding for analysis. This section details our data collection and data analysis steps and also discusses the validity threats.

3.1 Data Collection

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

We started with a study at Department A, whose results motivated the replication of a similar investigation at Department B. The research process is depicted in Figure 1.

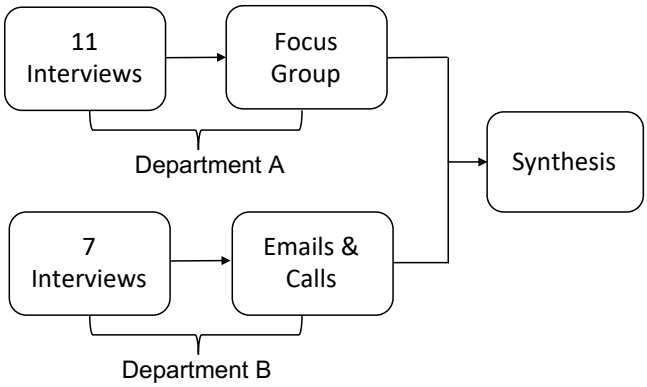

Figure 1. Research process for the two units of analysis

Semi-structured interviews were conducted at both departments with 11 interviews at Department A and followed up with a focus group meeting. The focus group meeting consisted of some of the interview participants and other new members. At Department B, seven interviews were conducted and followed up with emails and phone calls for failure to agree on a physical meeting time.

Throughout the different phases of this research, the authors collaborated in selecting participants and defining the interview guide based on consultations with our main contacts in both departments. Participants were selected from two role categories: a) the agile software teams and b) the plan-driven system level roles. Both categories think very differently about development: the agile software teams did not necessarily like to talk about requirements, while the system level roles did not feel that a discussion of agile artifacts would provide value. This was mainly a problem at Department A, where we chose to avoid using the word "requirements" in our interview guide for agile software teams, where necessary. In Department B, this aspect was deemed less critical, but since the way of working differed significantly, we had to adjust the interview guide [1] again for their context — allowing us to follow up on findings from Department A. We discuss each department in more detail in section 4.

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

3.2 Data Analysis

The interviews were transcribed and thematic coding method [12] was used to identify, analyze and report patterns within the data. Transcriptions were used to identify, name and categorize phrases and words in order to develop the initial codes. Interviewers also provided initial codes, enabling the generation of inductive codes which were later grouped into themes. The themes were iteratively and collaboratively refined by all authors and named according to the aspects addressed in the interview guide. We thus agreed on the following general themes; requirements engineering process (which included the teams involved and roles), challenges faced, and solutions proposed.

3.3 Threats to Validity

Three researchers worked iteratively with codes and their grouping into themes, during which potential misinterpretations were raised and discussed. In order to increase *conclusion validity*, the initial results were discussed in a focus group at Department A that included four roles already involved in the study and two more people responsible for requirements management. Results were presented in form of PowerPoint slides and discussed to check whether we misinterpreted any data and to provide an opportunity to raise any challenges we may have missed. We then discussed

[1] Interview guides: http://www.cse.chalmers.se/~knauss/2020-AgileIslands

Table 1. Roles and responsibilities of participants in the study.

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

strategies to overcome the challenges with the participants. For Department B, we shared a PDF version of the findings through email and when clarifications were needed, calls were made. To increase *internal validity*, we used data triangulation between the case departments. We take two departments within one company in an in-depth qualitative exploratory case study and through sufficient numbers of interviews and member checking, we made sure that we captured all important concepts in scope of our inquiry. Both cases however yield slightly different results as we will discuss in this paper. As a first exploration, we believe that this will enable future research in the area, and we deliberately choose a research method which sacrifices *generalization* of our findings to other domains in favour of in depth-knowledge in the specific case. We believe by discussing the results with company contacts in both departments and by comparing them to existing problems in literature, we mitigate threats to *reliability* of our findings.

4 The Case Study

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

In earlier work, system engineering organizations were found to differ in 'scope' as they introduce agile methods [14]. While few companies have achieved some level of agility for the full organization, at the time of this research, many had only introduced agile methods on the level of agile software development teams, while keeping a plan-driven approach for the overall systems engineering process. Our case company is no exception, yet both departments covered differ in how they manage the interplay of plan-driven systems engineering and agile software development. While both departments have significant experience with agile software development to support their business goal of increased flexibility, they differ in the type of software to be developed.

Department A is responsible for development of (often safety-critical) functions of the overall vehicle's system. Their development focuses on algorithms that connect large amounts of vehicle data with cloud intelligence, and agility helps to increase learning about how these algorithms behave in a realistic context early on(a comparable setup has been discussed in [16]). While suppliers for hardware, for example chip-sets exist, much of the software is done in-house to increase flexibility, decrease lead-time of new features, and protect innovations and intellectual property.

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

In order to give context to our findings, we extracted an overview of the current RE process at both departments, which we describe in terms of the roles and responsibilities (Tab. 1 and Fig. 2).

4.1 Roles in the Departments

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

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

The roles in the lower part in Table 1 relate to software development team as depicted at the bottom of Figure 2. Here the departments take different approaches to agility, (although both generally, use combinations of XP and Scrum in their development). Department A, has a Product Owner (PO) who is responsible for managing and prioritizing the backlog and then puts the requirements in Jira for implementation by the developers. In case of any change requests, the development team communicates to the PO who discusses with the FO to get the requirements changed. So the plan-driven requirements do not seem to directly interact with the agile process of development. In Department B, the SDE is the one that creates an *issue* in Jira which the developers use to implement the requirement. The software issue contains the 'user story' and the software requirement from SWRS. In case of any disagreements, the developers have to go through the SDE who then discusses with the FO, if necessary.

4.2 Requirements Model in the Departments

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

Figure 2 shows what our contacts call the *Narrow*-V model, the hierarchical model followed by both departments. It keeps the abstraction levels from the classical V-model [3] and has three general layers of requirements, but emphasizes iterative work on system level. On the left hand side, the requirements are formed and broken down at the different levels while the right hand side shows testers that write corresponding test cases for the requirements.

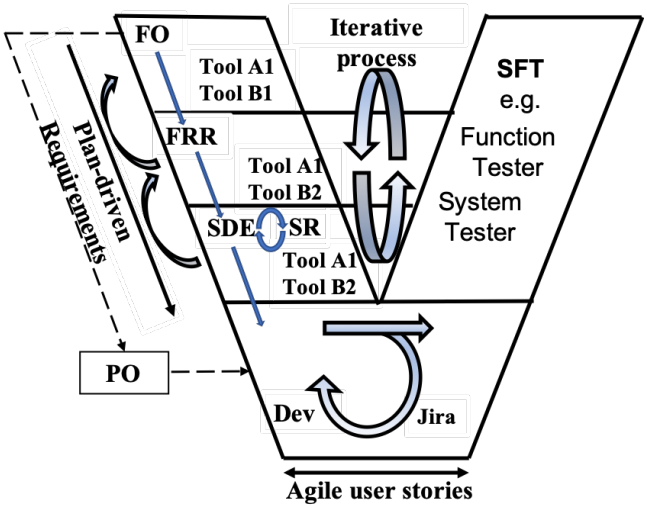

Figure 2. Requirements and user-story coexistence model.

The *Narrow*-V model is supported by different tools in the two departments. Department A uses Tool A1 for all the plan-driven levels while Department B uses two different tools; one for FO level (Tool B1) and another for FRR and SDE levels (Tool B2). For the agile part, the development part, both departments use the Jira tool to support their implementation. Anyone with access to Jira can write a requirement or user story to the developers in either case but the team only works on those approved by their immediate superiors, the PO for Department A and SDE for Department B.

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

5 Challenges and Strategies

This section describes the challenges and potential solutions identified through interviews and discussions in a focus group. These challenges are presented in three parts: In 5.1

the challenges common to both departments are discussed while 5.2 and 5.3 discuss those challenges unique to department A and B respectively. For each challenge, we discuss the corresponding potential strategies.

5.1 Challenges in Departments A and B

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

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

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

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

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

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

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

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

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

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

Both FOs in Department A however agree that there is a danger to become the bottleneck in this process.

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

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

5.1.3 C.3: 'Suffering' Requirements traceability.

The traceability between requirements artifacts used in the two departments is suffering, with differing levels of impact for each department. In Department A, the PO writes users stories from FO requirements and the team decides how to implement them. The team members can also decide to breakdown the stories to more granular levels and put to the backlog for discussion and later prioritisation. Clearly, not all user stories need to be traced, however, it is not clear which user stories should be traced to requirements. This

can lead to additional effort, when traceability must be ensured, since important information is missing and must be reconstructed. When asked to comment on traceability, one developer responded that:

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

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

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

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

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

5.2 Challenges Unique to Department A

5.2.1 C.4: All can write user stories anytime.

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

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

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

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

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

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

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

5.3 Challenges Unique to Department B

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

Proposed Solution: While this is a classical problem in software development, it becomes more pressing in iterative and agile development at scale. All interviewees agreed that maybe having a complete picture may not be practical but recommended visualisation of how each and every function is related to each other. A good first step is to make explicit how each part fits into the larger picture. This can for example be achieved by adding relevance for tracing explicit

for each backlog item (see C.3), or by making a formal test report part of hardening sprints before a release (C.1).

6 Discussion

Evidently, these two departments have adopted agile methods in different ways, and rely on different practices. Department B is more strict on user story management while Department A gives more freedom and responsibility to agile teams. Both departments embed agile teams in similar hierarchies defined from plan-driven systems engineering, but differences in the organizational interfaces affect the severity of requirements-related challenges faced. Department A teams have slightly more autonomy than teams in Department B and they frequently propose changes to FO. Figure 3 depicts the communication challenges introduced when departments define their own agile practices. Several studies [8, 13, 21] have mentioned and recommended organisations to tailor agility to their contexts. However, with agility being driven from the teams' perspective, we find that the tailoring is more on the team level with different teams having varying flavours of agile and yet they still need to contribute to the same product. We thus also notice a challenge at the inter-team level even though we did not specifically explore that direction and leave it for future work.

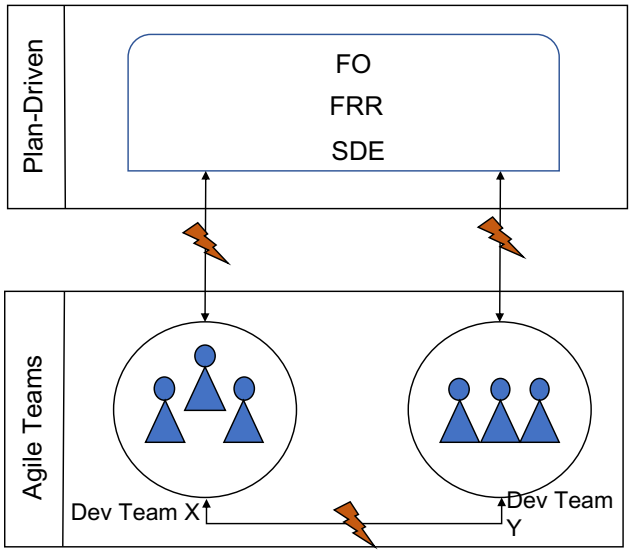

Figure 3. Challenges at the interfaces

Furthermore as shown in Figure 2, we noticed that Department A has one tool for requirements while Department B has several tools used by the different department roles. This diversity of tools makes it harder to see how the whole function operates or varies. Also, with many functions possibly connecting to other departments, the tools used in Department B make a difference in traceability and monitoring. Department A on the other hand seems to have less constraint on structures and thus face unique challenges which include: many user stories from different sources accessing the backlog and inconsistencies between the requirements model and implementation at testing time. These come up as consequence of less or not enough communication between testing and function owner team. On the other hand, Department B has teams that manage the requirements on a higher level and are thus able to monitor how the requirements change with respect to implementation. It is clear from the findings that the two departments have defined their own ways of working, which contributes to the difference in severity of challenges faced in the departments.

In Table 2 we give an overview of how our results relate to related work. The cross functional teams have been proposed in literature to handle challenges C.1 and C.2. For this the teams would have to comprise of the required expertise to handle the update issues to FO and also manage high-level requirements within the team. For C.3 where it is not clear whose responsibility traceability is, Liebel *et al.*[22] recommend increased understanding of processes and roles to address the challenge of 'unclear responsibilities and borders'. It is worthy noting that these challenges could also be instigated by the failure to implement the agile practices well. Kuusinen *et al.*[21] recognise that as a challenge and recommend co-located teams and having a permanent workforce. In our opinion, this recommendation does not match our case context as the teams are already co-located and permanent to the extent to which it is possible to do so at this scale.

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

7 Conclusion

In conclusion, we present results from the interplay of agile teams and traditional structures in software development. By studying the requirements flow with focus on the roles and structures used for requirements communication, we were able to identify challenges brought about by such coexistence. In both departments, and basing on the challenges observed, the current state of the coexistence of agile and plan-driven approaches seems to limit the efficiency of development. The departments have different ways of working and use different tools. Each team implements agile methods in their own way, which leads to some differences in the challenges we observed. Generally, all challenges relate to working with requirements updates when the team uses agile methods while the department structure is plan-driven. Strategies, in the literature, to mitigate these challenges range from having cross-functional teams, improving traceability through adequate tooling to improving process understanding and having well defined roles that may include empowering teams to manage requirements on their end. We believe that for

Table 2. Summary of Challenges and Proposed Strategies

Challenge	Strategies	
	Interviewees	Literature
C.1 Development team unaware of requirement model	Have testers as part of agile team Team creates formal test report for release	Cross-functional teams [5, 22]
C.2 FO over exposed to change requests	(Empower team to) update requirements based on learning from sprint	Cross-functional teams for requirements update [5, 17]
C.3 Suffering requirements traceability	Explicitly define if user story must be traced Force team to understand reqts. model	Increase understanding of process and roles [22]
C.4 All can write user stories anytime	Create proper channels for writing user stories	Cross-functional teams update requirements and peer-review [17]
C.5 Inconsistency between requirements model and user stories implementation	Have testers as part of team	Bring testers closer to requirement owner [30] Iterative requirements management [10]
C.6 Complete picture and system thinking missing	Visualisation of how functions relate	

agile automotive system development, a healthy way must be found that empowers teams to take more responsibility for high-level requirements, without sacrificing too much of the agile spirit. Practitioners in similar settings can use our findings to facilitate process improvement and to drive agile transformations. Future research will have to show to which extent these or other solutions can mitigate the challenges and to get most out of the combination of agile and plan-driven approaches.

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

Acknowledgments

We thank all participants in this study for their great support, deep discussions, and clarifications. Special thanks goes to Swathi for her help with data collection. This work was supported by Software Center Project 27 on RE for Large-Scale Agile System Dev. and the Sida/BRIGHT project 317 under the Makerere-Swedish bilateral research programme 2015-2020.

References

[1] Agile Alliance. 2019. Glossary: Scrum Master. https://www.agilealliance.org/glossary/scrum-master/ last visit: 2019-Oct-18.

[2] Fernando Almeida. 2017. Challenges in Migration from Waterfall to Agile Environments. *World Journal of Computer Application and Technology* 5, 3 (2017), 39–49.

[3] S Balaji and M Sundararajan Murugaiyan. 2012. Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. *International Journal of Information Technology and Business Management* 2, 1 (2012), 26–30.

[4] Sander Bannink. 2014. Challenges in the Transition from Waterfall to Scrum–a Casestudy at Portbase. In *20th Twente Student Conference on Information Technology*. University of Twente, Netherlands.

[5] Elizabeth Bjarnason, Krzysztof Wnuk, and Björn Regnell. 2011. A case study on benefits and side-effects of agile practices in large-scale requirements engineering. In *Proceedings of 1st Workshop on Agile Requirements Engineering*. ACM, Lancaster, UK, 3.

[6] Barry Boehm and Richard Turner. 2005. Management challenges to implementing agile processes in traditional development organizations. *IEEE software* 22, 5 (2005), 30–39.

[7] Alessio Bucaioni, Antonio Cicchetti, Federico Ciccozzi, Manvisha Kodali, and Mikael Sjödin. 2018. Alignment of Requirements and Testing in Agile: An Industrial Experience. In *Information Technology - New Generations*, Shahram Latifi (Ed.). Springer International Publishing, Cham, 225–232.

[8] Amadeu Silveira Campanelli and Fernando Silva Parreiras. 2015. Agile methods tailoring – A systematic literature review. *Journal of Systems and Software* 110 (2015), 85 – 100. https://doi.org/10.1016/j.jss.2015.08.035

[9] Ronald S Carson. 2013. 4.2. 1 Can Systems Engineering be Agile? Development Lifecycles for Systems, Hardware, and Software. In *INCOSE International Symposium*, Vol. 23. Wiley Online Library, Philadelphia, PA, 16–28.

[10] Francisco Gomes de Oliveira Neto, Jennifer Horkoff, Eric Knauss, R. Kasauli, and G. Liebel. 2017. Challenges of Aligning Requirements Engineering and System Testing in Large-Scale Agile: A Multiple Case Study. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*. IEEE, Lisbon, Portugal, 315–322. https://doi.org/10.1109/REW.2017.33

[11] Kim Dikert, Maria Paasivaara, and Casper Lassenius. 2016. Challenges and Success Factors for Large-Scale Agile Transformations: A Systematic Literature Review. *Journal of Systems and Software* 119 (2016), 87–108.

[12] Graham R Gibbs. 2008. *Analysing qualitative data* (2nd ed.). Sage, London, UK.

[13] Georg Kalus and Marco Kuhrmann. 2013. Criteria for Software Process Tailoring: A Systematic Review. In *Proceedings of the 2013 International Conference on Software and System Process (ICSSP 2013)*. Association for Computing Machinery, New York, NY, USA, 171–180. https://doi.org/10.1145/2486046.2486078

[14] R. Kasauli, G. Liebel, E. Knauss, S. Gopakumar, and B. Kanagwa. 2017. Requirements Engineering Challenges in Large-Scale Agile System

Development. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE, Lisbon, Portugal, 352–361. https://doi.org/10.1109/RE.2017.60

[15] Carine Khalil. 2018. The State of the Practice of Agile and Plan-Driven Approaches in ICT Development Projects: An Exploratory Research Study. In *Digital Technology and Organizational Change*. Springer, Verona, Italy, 25–33.

[16] Eric Knauss, Andreas Andersson, Michael Rybacki, and Erik Israelsson. 2015. Research Preview: Supporting Requirements Feedback Flows in Iterative System Development. In *Proceedings of 21st International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ? 15)*. Springer, Cham, Essen, Germany, 277–283.

[17] Eric Knauss, Grischa Liebel, Jennifer Horkoff, Rebekka Wohlrab, Rashidah Kasauli, Filip Lange, and Pierre Gildert. 2018. T-Reqs: Tool support for managing requirements in large-scale agile system development. In *2018 IEEE 26th International Requirements Engineering Conference (RE)*. IEEE, Banff, Alberta, Canada, 502–503.

[18] Marco Kuhrmann, Philipp Diebold, Jürgen Münch, Paolo Tell, Vahid Garousi, Michael Felderer, Kitija Trektere, Fergal McCaffery, Oliver Linssen, Eckhart Hanser, and et al. 2017. Hybrid Software and System Development in Practice: Waterfall, Scrum, and Beyond. In *Proceedings of the 2017 International Conference on Software and System Process (ICSSP 2017)*. Association for Computing Machinery, Paris, France, 30–39. https://doi.org/10.1145/3084100.3084104

[19] M. Kuhrmann, P. Diebold, J. Munch, P. Tell, K. Trektere, F. McCaffery, V. Garousi, M. Felderer, O. Linssen, E. Hanser, and C. R. Prause. 2019. Hybrid Software Development Approaches in Practice: A European Perspective. *IEEE Software* 36, 4 (July 2019), 20–31. https://doi.org/10.1109/MS.2018.110161245

[20] Rob J Kusters, Youri van de Leur, Werner GMM Rutten, and Jos JM Trienekens. 2017. When Agile Meets Waterfall - Investigating Risks and Problems on the Interface between Agile and Traditional Software Development in a Hybrid Development Organization.. In *Proceedings of the 19th International Conference on Enterprise Information Systems*, Vol. 2. SCITEPRESS-Science and Technology Publications, Lda., Porto, Portugal, 271–278.

[21] Kati Kuusinen, Peggy Gregory, Helen Sharp, and Leonor Barroca. 2016. Strategies for Doing Agile in a Non-Agile Environment. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '16)*. Association for Computing Machinery, New York, NY, USA, Article Article 5, 6 pages. https://doi.org/10.1145/2961111.2962623

[22] Grischa Liebel, Matthias Tichy, Eric Knauss, Oscar Ljungkrantz, and Gerald Stieglbauer. 2018. Organisation and communication problems in automotive requirements engineering. *Requirements Engineering* 23, 1 (2018), 145–167.

[23] Mikael Lindvall, Dirk Muthig, Aldo Dagnino, Christina Wallin, Michael Stupperich, David Kiefer, John May, and Tuomo Kahkonen. 2004. Agile software development in large organizations. *Computer* 37, 12 (2004), 26–34.

[24] Bertrand Meyer. 2014. *Agile! The Good, the Hype and the Ugly*. Springer Internation Pulishing, Switzerland.

[25] Joakim Pernstål, Ana Magazinius, and Tony Gorschek. 2012. A study investigating challenges in the interface between product development and manufacturing in the development of software-intensive automotive systems. *International Journal of Software Engineering and Knowledge Management* 22, 07 (2012), 965–1004.

[26] Per Runeson, Martin Höst, Austen Rainer, and Björn Regnell. 2012. *Case study research in software engineering: Guidelines and examples* (1 ed.). John Wiley & Sons, Hokoben, New Jersey.

[27] Alberto Sillitti and Giancarlo Succi. 2005. Requirements engineering for agile methods. In *Engineering and Managing Software Requirements*. Springer, Berlin, Heidelberg, 309–326.

[28] Sven Theobald and Philipp Diebold. 2018. Interface Problems of Agile in a Non-agile Environment. In *International Conference on Agile Software Development*. Springer, Porto, Portugal, 123–130.

[29] Georgios Theocharis, Marco Kuhrmann, Jürgen Münch, and Philipp Diebold. 2015. Is water-scrum-fall reality? on the use of agile and traditional development practices. In *PROFES*. Springer, Bolzano, Italy, 149–166.

[30] E. J. Uusitalo, M. Komssi, M. Kauppinen, and A. M. Davis. 2008. Linking Requirements and Testing in Practice. In *2008 16th IEEE International Requirements Engineering Conference*. IEEE, Barcelona, Catalunya, Spain, 265–270. https://doi.org/10.1109/RE.2008.30

[31] Guus Van Waardenburg and Hans Van Vliet. 2013. When agile meets the enterprise. *Information and software technology* 55, 12 (2013), 2154–2171.

[32] Rebekka Wohlrab, Patrizio Pelliccione, Eric Knauss, and Sarah C Gregory. 2018. The Problem of Consolidating RE Practices at Scale: An Ethnographic Study. In *Requirements Engineering: Foundation for Software Quality*. Springer International Publishing, Cham, 155–170.

[33] Robert K Yin. 2009. *Case study research: Design and methods; 4th ed.* Sage, London.