Empir Software Eng (2017) 22:2892–2936
DOI 10.1007/s10664-016-9491-z

Managing the requirements flow from strategy to release in large-scale agile development: a case study at Ericsson

Ville T. Heikkilä[1] · **Maria Paasivaara**[1] · **Casper Lasssenius**[1] · **Daniela Damian**[2] · **Christian Engblom**[3]

Published online: 10 January 2017

Abstract In a large organization, informal communication and simple backlogs are not sufficient for the management of requirements and development work. Many large organizations are struggling to successfully adopt agile methods, but there is still little scientific knowledge on requirements management in large-scale agile development organizations. We present an in-depth study of an Ericsson telecommunications node development organization which employs a large scale agile method to develop telecommunications system software. We describe how the requirements flow from strategy to release, and related benefits and problems. Data was collected by 43 interviews, which were analyzed qualitatively. The requirements management was done in three different processes, each of which had a different process model, purpose and planning horizon. The release project management process was plan-driven, feature development process was continuous and implementation management process was agile. The perceived benefits

Communicated by: Daniel Amyot

✉ Ville T. Heikkilä
ville.t.heikkila@aalto.fi

Maria Paasivaara
maria.paasivaara@aalto.fi

Casper Lasssenius
casper.lassenius@aalto.fi

Daniela Damian
damian.daniela@gmail.com

Christian Engblom
christian.engblom@ericsson.com

[1] Aalto University, PO Box 15400, FI-00076, Aalto, Finland

[2] University of Victoria, PO Box 1700, STN CSC, Victoria, BC V8W 2Y2, Canada

[3] Oy LM Ericsson AB, Kirkkonummi, Finland

included reduced development lead time, increased flexibility, increased planning efficiency, increased developer motivation and improved communication effectiveness. The recognized problems included difficulties in balancing planning effort, overcommitment, insufficient understanding of the development team autonomy, defining the product owner role, balancing team specialization, organizing system-level work and growing technical debt. The study indicates that agile development methods can be successfully employed in organizations where the higher level planning processes are not agile. Combining agile methods with a flexible feature development process can bring many benefits, but large-scale software development seems to require specialist roles and significant coordination effort.

Keywords Scaling agile software development · Requirements management · Scrum · Large projects · Telecommunications software

1 Introduction

The traditional, plan-driven product and project management models are not well suited for agile development organizations where scoping decisions must be made frequently and requirements engineering is performed concurrently with implementation (Jantunen et al. 2011). If the requirements management processes do not support the agile development organization, it is difficult for the development organization to work efficiently towards the high level goals of the company. Due to the short history of agile methods use in large organizations, reports on the best practices of agile development in large organizations are lacking and many large organizations are struggling to implement efficient requirements processes (Laanti et al. 2011; Wiklund et al. 2013; Cao et al. 2004). Although there is an increasing number of empirical studies of large-scale agile development (e.g., Korhonen 2013; Laanti et al. 2011; Heikkilä et al. 2015b; Moe et al. 2014; Bass 2015; Eckstein 2014), there is little research on requirements management in large-scale agile development organizations (Heikkilä et al. 2015a; Inayat et al. 2015). Furthermore, most of the existing empirical research has focused on method proposal and evaluation instead of understanding the phenomenon (Heikkilä et al. 2015a). Subsequently, more research is warranted in order to identify the contextual factors that affect the success or failure of specific ways of requirements management in large organizations that employ agile development methods. Moreover, requirements engineering activities are complex and intertwined with other development and management processes in the organization (Damian and Chisan 2006), equally affected by human, organizational and political aspects that surround them (Maiden 2012). Furthermore, detailed information on requirements engineering practice in large organizations, in general, is scarce (Maiden 2012).

Our goal is to describe the requirements processes on the release and implementation management levels, and the interactions between the levels in a large organization that develops telecommunications network software and uses agile practices in its software development. We aim to reach this goal by studying the case organization and answering the following research questions:

RQ1: How is the requirements flow from the strategy to a release managed?

RQ1.1: What are the organizational roles involved in the requirements flow?

RQ1.2: What are the processes of the requirements flow?

RQ2: What are the perceived benefits of the requirements management processes?

RQ3: What are the perceived problems related to the requirements management processes?

The main contribution of this research is the in-depth description of these management processes in the case organization. The existing literature on requirements engineering in agile development is largely based on the single-team, single customer context (Heikkilä et al. 2015a; Inayat et al. 2015). To the best of our knowledge, our work is among the first to uncover requirements engineering practices as embedded through the feature development as well as the release project management and implementation management processes of a large-scale agile development organization. Those aspects of large-scale agile development that are not directly related to requirements management are out of the scope of our research. These include, but are not limited to, communication tools, coaching, continuous improvement, agile culture and agile contracts.

In a previous publication, we gave a preliminary description of how release planning was conducted in the case organization (Heikkilä et al. 2013b). This paper considerably expands that publication both by scope and by depth and also contains data from four additional interviews. The scope of this paper is expanded to include the interfaces between the different management levels, and it focuses on the requirements engineering practices that are embedded within these levels. We provide an in-depth description of the actors, artifacts and processes involved in the management of requirements and release projects, and analyze both new and previously identified problems and benefits in more detail than in our previous publication.

We have also studied the implementation planning process of the case organization (Heikkilä et al. 2013a), identifying quantitatively and explaining qualitatively the discrepancies between the implementation planning process of the development teams in the case organization and the normative Scrum process (Schwaber and Beedle 2002). Paasivaara et al. (2013) described how global sites were included in the agile transformation at Ericsson. Paasivaara et al. (2014) studied how defining common values supported the agile transformation at Ericsson. The focus and goals of this paper are considerably different from these previous publications.

The rest of the paper is organized as follows: Section 2 discusses related work in order to provide background knowledge and position our study. Section 3 describes our data collection and data analysis methods. Section 4 provides background information on the case organization. Section 5 presents the findings of the study. The findings, limitations and threats to validity are discussed in Section 6. Finally, Section 7 concludes the paper and gives directions for future work.

2 Background and Related Work

In this section, we review related work in order to position our research in the field of requirements management and software engineering research. We also present background information that is beneficial for understanding our case study and its relation to previous research. First, we summarize two recent secondary studies on agile requirements engineering. Second, we discuss research on organizing and managing large-scale agile development. Third, we review three models proposed for scaling agile development in order to provide a point of comparison to our case.

2.1 Agile Requirements Engineering

Secondary studies on agile requirements engineering have been recently conducted by Inayat et al. (2015) and Heikkilä et al. (2015a). Their findings are summarized below.

There is no universally accepted definition of agile requirements engineering (agile RE) (Heikkilä et al. 2015a). Inayat et al. (2015) identified the following 17 RE practices that were adopted in agile software development: Face-to-face communication between the agile team members and client representatives, customer involvement and interaction, user stories as specifications of the customer requirements, iterative requirements that emerge over time, iterative requirements prioritization, challenge management, cross-functional teams, prototyping, test-driven development, requirements modeling, requirements management with a product backlog, review meetings and acceptance tests, code refactoring, shared conceptualizations, retrospectives, continuous planning and pairing for requirements analysis. According to Heikkilä et al. (2015a), the activities performed in agile and traditional RE have similar goals, but the methods and emphases are different, since traditional RE emphasizes processes and documents while agile RE emphasizes reactivity and informal communication.

Both articles identified benefits that agile RE has been claimed to have over traditional RE (Heikkilä et al. 2015a; Inayat et al. 2015). Agile RE is claimed to decrease process overheads[1] due to the smaller amount of required requirements and system documentation. The frequent requirements and system validation by the customer(s) and the frequent face-to-face communication are claimed to improve the understanding about requirements and prevent communication gaps. Agile RE practices are claimed to reduce overallocation of development resources. Agile RE is claimed to be more responsive to changes in the environment or in the customers' needs. Finally, agile RE is claimed to improve customer relationships.

Both articles also identify challenges in the agile RE approach (Heikkilä et al. 2015a; Inayat et al. 2015). Agile RE relies on the availability of the customers, but due to cost, time and trust issues the access to the customers or customer representatives is often limited. Furthermore, the customers or the customer representatives may have conflicting needs, they may be unwilling to prioritize requirements or they may not accurately represent the needs of the customers' organizations. The reliance on the simple user story format for requirements documentation is problematic when requirements need to be communicated to off-site stakeholders and the user story format may be insufficient for complex, large-scale systems development. Since most requirements knowledge is tacit in agile RE, personnel turnover is problematic. Non-functional requirements and system improvements may be understated due to the customer value emphasis of agile RE. The de-emphasis of planning and the short planning time horizon in agile RE may result in an inappropriate architecture and technical debt. Precise budget and schedule estimation required by development contracts is difficult without extensive planning, but due to the volatility of agile RE, extensive planning is not considered worthwhile.

Solutions to the aforementioned problems are discussed in the articles (Heikkilä et al. 2015a; Inayat et al. 2015), but most solutions are on the level of a proposal and the validation

[1] In this paper, overhead refers to the effort spent on work that does not directly contribute to the end product, but is necessary nevertheless. For example, learning or creating infrastructure are usually considered overhead when learning or infrastructure are not the end product.

of the proposed solutions is lacking (Heikkilä et al. 2015a). Typically, the proposed solutions are based on the re-introduction of traditional RE practices, roles or artifacts.

2.2 Planning in Large-Scale Agile Development

One way to scale an agile development organization is to employ multiple small teams that collaborate and share a common goal (Leffingwell 2011; Schwaber 2007; Augustine 2008). In a such organization, the product roadmaps are agnostic towards the development methodology (Lehtola et al. 2005) and the development teams can employ any suitable agile method. However, the release planning process must support the agile development teams by providing goals and direction on what should be constructed (Rautiainen et al. 2002) and by assisting in the inter-team coordination (Maglyas et al. 2012). On the other hand, the release management process must take into account the realized development progress and communicate it to the strategic management in order to give a realistic picture of the status of the software development.

There is some evidence that adoptions to agile development life-cycle models must be made in order to make them work well in a large-scale, multi-team development organization. Cao et al. (2004) reported changes to Extreme Programming (Beck and Andres 2004) that were made to adopt it to a large-scale development organization in a financial application development context. The system architecture was planned six-months up-front, instead of expecting the architecture to emerge during the development and the developers employed a limited number of predefined design patterns, instead of designing and developing everything from scratch. Heikkilä et al. (2015a) found that large organizations developing complex software systems cannot rely solely on face to face communication and simple, user story based requirements documentation when they adopt agile development methods.

In consumer market software product development, the success of the product is tied to the completion of the right set of requirements by the time of the public release (Svahnberg et al. 2010; Fogelström et al. 2010; Chow and Cao 2008) and the release schedule may be tied to dates mandated by trade shows, holiday seasons or competitors' releases, for example. Software releases in telecommunications network software development have traditionally been sparser due to the high fixed cost often associated with version updates, and due to a risk averse attitude that follows from the mission critical nature of lots of the software. However, the ability to respond to customer requests for new or improved functionality in a timely fashion creates a competitive advantage also in the telecommunications network software development context. Furthermore, the recent rise of software-defined telecommunications networks emphasizes the importance of the software side of the networks development (Batista et al. 2015).

There is a notable lack of empirical research on requirements management in large-scale agile organizations that develop large, complex and hardware dependent software systems such as telecommunications network software. In contrast to consumer market applications, telecommunications software controls devices and mostly communicates with other devices or software systems instead of a human user (Taramaa et al. 1996; Lee 2002). The software development organization in such an environment is often an internal producer for the more extensive systems development organization and the software is only one part of the system or service that is provided for the customers. Unlike in consumer market software product development, the requirements for the telecommunications network software stem from a wide variety of sources in the encompassing systems development organization. In addition, telecommunications network software often requires customization in

order to fit the customers' environment due to technical and regulatory reasons. These aspects make telecommunications network software development inherently different from consumer market application development.

2.3 Large-Scale Agile Development Organization Models

Although research literature on large-scale agile development organizational models is scarce, many consultants and practitioners have proposed different kinds of models for scaling agile development based on their experiences. We briefly review three notably different prescriptive models to provide an overview of the subject and to allow comparisons to our case. The model proposed by Schwaber (2007) has been included because Ken Schwaber is considered the most influential figure in the creation of the original Scrum method (Schwaber and Beedle 2002). At the time of writing, two popular commercial scaled agile frameworks are Large-Scale Scrum (LeSS)[2] and Scaled Agile Framework (SAFe)[3], both being prescriptive models. Figure 1 illustrates the different proposed development organizations.

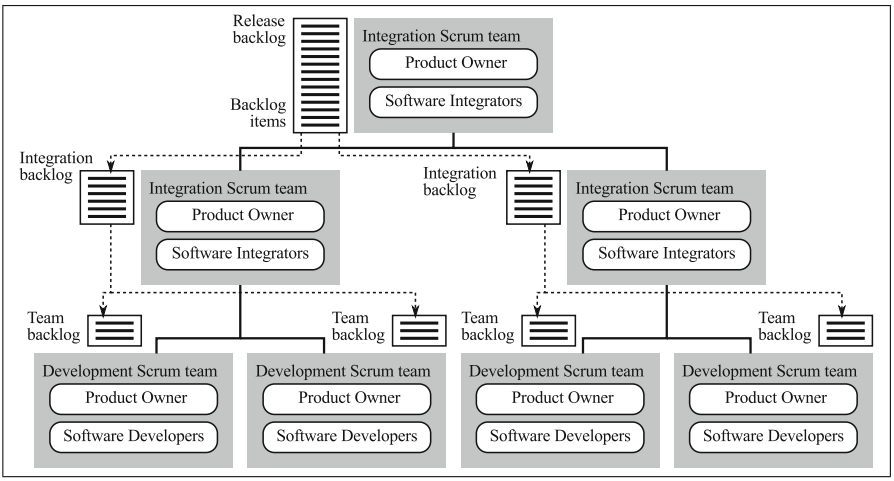
Schwaber (2007) suggests organizing development using a tree structure of multiple levels of integration Scrum teams in the branch nodes and (development) Scrum teams in the leaf nodes. The integration Scrum teams do not develop functional software, but instead integrate, build and test the software implemented by the (development) Scrum teams. Both kinds of Scrum teams have a dedicated product owner. All requirements are listed in a product backlog as user stories. The branch node product owners are responsible for assigning sections of the product backlog to the lower level teams. Release planning is performed by the root node product owner by selecting a subset of the product backlog as the release product backlog.

Larman and Vodde (2010) propose a two-layer model for scaling a large-scale development organization. The further elaborations and extensions of this model have been commercialized in the Large-Scale Scrum (LeSS) framework. Development teams are arranged as feature teams that work on a single feature at a time and the team composition persists over time. Feature teams are grouped into technical product areas. Each product area is managed by an area product owner, who in turn are managed by a product owner. The product owner manages the product backlog and assigns backlog items to the product areas. Features are large backlog items that describe functionality that is valuable for the customer. Features are split into smaller backlog items which can be implemented during a single sprint. Only the dates of the external releases are planned, and the content of the release is defined by what is ready at the time of the release.

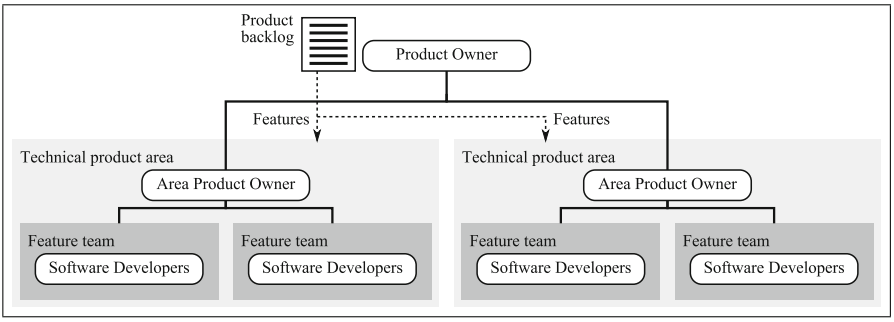
Leffingwell (2011) suggests a three-layer model of the agile enterprise. A more detailed version of this framework has been commercialized as the Scaled Agile Framework (SAFe). The three layers are the portfolio, the program and the team layer. The portfolio layer is planned with epics which are "large-scale development initiatives" that span multiple releases and that are stored in the epic backlog. Epics are split into features which are planned on the program layer and stored in the feature backlog. Features are descriptions of system behavior that can be developed in a single release project. Product management is responsible for managing the program backlog which contains the Features. Features are split into stories which can be implemented in a single development Iteration. The

[2]See http://less.works/

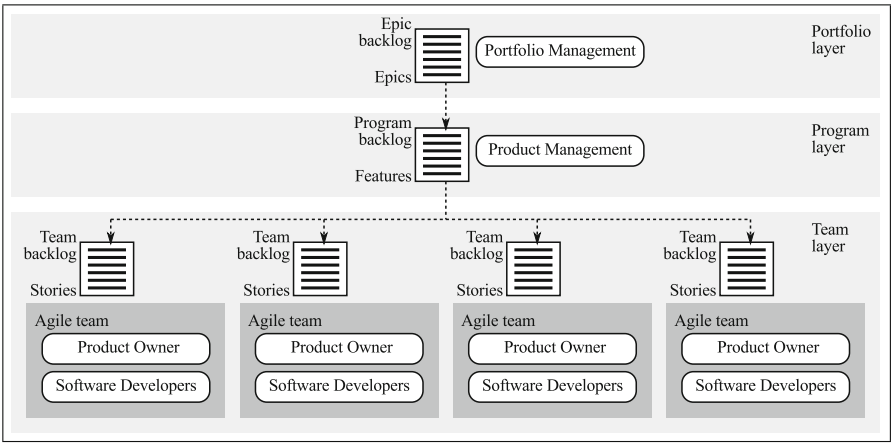
[3]See http://www.scaledagileframework.com/



(a) Schwaber (2007)



(b) Larman and Vodde (2010)



(c) Leffingwell (2011)

Fig. 1 Organization of large-scale agile development by (**a**) Schwaber, (**b**) Larman and Vodde and (**c**) Leffingwell (figures originally from Heikkilä (2015) reproduced with permission)

developers are organized in independent teams that each have a dedicated product owner. Release planning is performed in release planning events where all stakeholders of the product assemble to plan the next release together. A case study of release planning using Leffingwell's ideas has been published by Heikkilä et al. (2015b).

Although these models are purportedly based on experiences in real software development organizations, the empirical validation of the models is weak. As Fig. 1 illustrates, the structure of the organization differs between the models considerably, as do the requirements management processes. Clearly more empirical research on planning, organizing development and managing requirements is required to evaluate the benefits and problems of different agile scaling models, as well as to study to what kind of circumstances each is best suitable.

3 Research Methods

This study employed the case study method (Yin 2009), which is the most appropriate when a contemporary phenomenon is studied in its real-life context (Yin 2009), as was the case in our study. Data was collected with interviews. The data was analyzed with the qualitative content analysis approach (Patton 2002; Hsieh and Shannon 2005; Elo and Kyngäs 2008). Below, we first describe our data collection in detail. Second, we describe how the data was analyzed.

3.1 Data Collection

Ericsson was purposefully selected as the subject of our case study, as it provided an opportunity to perform an information rich study (Patton 2002; Yin 2009) in a large organization with a long history of developing a complex software system. The company was globally distributed and developed large telecommunications infrastructure systems that consisted of both software and hardware. Our study was focused on the software development in an Ericsson telecommunications node development organization.

We conducted three rounds of interviews. Details of the data collection are shown in Table 1. The two first interview rounds were conducted in spring 2011 and the third round in spring 2013. Eight interviewees were selected for the first interview round with the help of a case organization representative in order to get a good overview of the organization. These eight interviewees were our *key informants* (Yin 2009). They were selected based on their long software engineering experience, experience in the case organization and based

Table 1 Details of the data collection

Interview round	Focus	Interviewees	Roles
1. (Mar 2011)	Organization and process overview	Finland: 8	6 Managers, Chief Product Owner, Scrum Master
2. (Jun 2011)	Deeper understanding of the management processes	Finland: 19, Hungary: 11	13 Development team members, 2 Managers, 7 Product Owners, 5 Technical specialists, 5 Scrum Masters*
3. (Apr 2013)	Development teams' operational management processes	Finland: 5	Product Owner, 4 Developers**

on their ability to provide an overview of the organization history, goals, growth, structure and the requirements management processes used in the organization. The details of the key informants' roles, software engineering experience and backgrounds are shown on Table 2. To build deeper understanding of the case organization and to enable the triangulation of data sources (Patton 2002), we performed the second round of interviews by interviewing 19 persons in Finland and 11 persons in Hungary. The goal of this second interview round was to interview multiple people, if available, with the same role in the organization in order to triangulate data sources (Yin 2009; Patton 2002) and improve construct validity. Almost all interviews were conducted by two interviewers in co-operation. After a careful analysis of the data from the first two interviews rounds, we found that we needed to deepen our understanding of the team-level practices in the case organization and decided to perform an additional interview round. This third round was focused on how the development teams managed requirements. We interviewed one product owner and four developers from three different development teams in Finland.

We selected the *general interview guide* approach (Patton 2002) in order to maintain adaptability to the roles and individual experiences of the interviewees while simultaneously making sure that the relevant topics were explored. We updated the interview guide constantly based on new insights from the previous interviews (Patton 2002). An overview of the questions asked in the interviews can be found in Appendix. We began each interview by explaining who we were and what was the purpose of the interview. Then we asked the interviewee to describe his or her history with the company and the current role in the organization. The rest of the questions asked from each interviewee were based on the role of the interviewee and on the subjects we wanted to know more about. Thus, the set of questions asked from each interviewee differed somewhat. Most interviews were conducted by two interviewers. One interviewer asked most of the questions and the other took detailed notes and asked additional questions whenever he or she thought that the some topic needed additional information. All interviews were voice-recorded and extensive notes on the questions and answers were taken during the interviews.

Table 2 Details of the key informants

Current role	Experience	Work history
Development process manager	$>$ 30 years	Software development, testing, product management, line management
Portfolio manager	$\approx$ 20 years	Software development, project management, line management
Product manager	$\approx$ 25 years	Product marketing, product management
Continuous integration manager	$\approx$ 10 years	Testing, test management
Chief Product Owner	$\approx$ 20 years	Software architect, project management, line management, testing
Site manager	$\approx$ 17 years	Software development, system architect, project management, line management
Program manager	$\approx$ 10 years	Software development, QA manager, project management, line management
Scrum Master	$\approx$ 20 years	Software development, project management, QA manager

3.2 Data Analysis

The first, informal steps in the analysis process were taken already during the interviews when we decided which questions to ask from each interviewee based on the previous answers and interviews. Since we used the general interview guide approach (Patton 2002), these decisions were made on the spot.

The interview records were transcribed by a professional transcription company. We analyzed the interviews with the inductive qualitative content analysis method (Patton 2002; Hsieh and Shannon 2005; Elo and Kyngäs 2008). The qualitative content analysis method aims to identify core consistencies and meanings in qualitative data. We took the inductive analysis approach in order to avoid forcing the data into any preconceived conceptual framework or theory. The inductive approach is the most suitable when there is no former knowledge on the topic of the study or the knowledge is fragmented (Elo and Kyngäs 2008). Based on secondary studies by Heikkilä et al. (2015a); Inayat et al. (2015), this indeed is the state of knowledge regarding requirements management in large-scale agile development organizations.

The transcripts were imported into Atlas.ti, which is a qualitative data analysis program. The research questions were used to sensitized us to identify passages of text that were relevant to our goal. These included passages that somehow described the requirements flow (RQ1), passages where the interviewee described a benefit in the requirements management processes (RQ2) and passages where the interviewee described a problem in the requirements management processes (RQ3). We coded the transcripts using the constant comparison method: We begun to read from the beginning of the first transcript and identify potential codes based on passages of text, which were then coded using the identified codes. As we continued to read the transcripts, we compared new passages of text with the existing codes. A passage could then either indicate a new code, reinforce an existing code or indicate that an existing code does not describe the phenomenon very well and a new code needs to be identified to replace it.

As the number of codes increased, we began to identify potential categories that encompassed multiple codes. Using the constant comparison method, the categories were constantly reviewed based on codes and altered, combined or removed if the data did not support the categories. The categories had three functions. They helped to keep the code book in order and within a manageable size by grouping similar codes that could be potentially combined, they gave the codes additional meaning (such as the category Work Item giving meaning to the code Epic) and they sensitized us to identify new codes that could potentially fit the existing categories. The categories and concepts identified in the analysis are shown in Table 3. We coded a total of 625 passages from the first and second round interviews and 79 passages from the third round interviews.

The construction of the case description was an iterative process. We extracted passages of text related to a code or a combination of codes of interest. The extracted passages were read and the description of the case was augmented with the information from the excerpts. During the process, new queries were constructed in order to clarify or extend parts of the case description. For example, we identified an organizational unit that was called the product owner team. In order to construct the description of the product owner team, we extracted passages of text coded with “chief product owner” OR “product owner” and re-read them.

Figure 2 illustrates, as an example, how the description of One Pager was constructed from the excerpts. In total, 42 passages of text were coded with the code “One Pager” and these all contributed to the description of how the One Pagers were created, processed and

Table 3 Categories and codes that were identified in the interview coding process

Category	Definition	Codes
Action	Actions that alter a requirement or a property of a requirement, for example the splitting or prioritization of a requirement.	Assigning/selecting work items, Estimating work items, Prioritizing work items, Splitting work items
Planning artifact	Artifacts that are created and/or employed in the planning processes.	Backlog, Feature concept study, One pager, Requirements specification, Roadmap
Planning horizon	Distinct stretches of the temporal dimension that have a specific purpose in the planning processes.	Daily, Release/project, Strategic
Process description	Events or properties of events that have a distinct purpose in the planning processes.	Backlog grooming, Early phases, F-Decisions, P-Decisions, Portfolio meeting, Quality status meeting, Scrum of Scrums, Sprint planning, Sprint review/retrospective, Status meeting, Steering group, Tollgate decision
Stakeholder	Roles or organizational units that affect or are affected by the planning processes.	Capability management, Development team, Early phases program, Line organization, Node architect, Chief product owner, Portfolio management, Product owner, Product management, Program manager, Project manager, Release management, Scrum master, Technical management
Work item	Embody work that needs to be completed, for example a requirement or a bug report.	Change request, Epic, Feature, Requirement, Bug report, User story
Attitude	Expressions of attitudes towards something.	Benefit, Problem, Motive

used. Section 5 contains several excerpts from the interviews. The specific excerpts are included only as exemplars of the data that was used to construct the section.

4 The Case Organization

The studied Ericsson node development unit develops a large systems product consisting of both software and hardware, responsible for handling a specific type of traffic in telecommunications networks. In 2013, the development of this product had been going on for over twelve years and further development of the product still continued. The product is used by over 300 telecommunications operators all over the world.

The organization begun a process improvement initiative in 2009. The existing, plan-driven process worked quite well, but the management wanted to decrease the requirements development lead time, improve flexibility and increase the motivation of the developers. The management studied different options and initially chose Scrum as the best fit for the organization's needs. They also studied the agile scaling ideas proposed by Larman and Vodde (2009). Thus, the scaling practices were inspired by the ideas that were

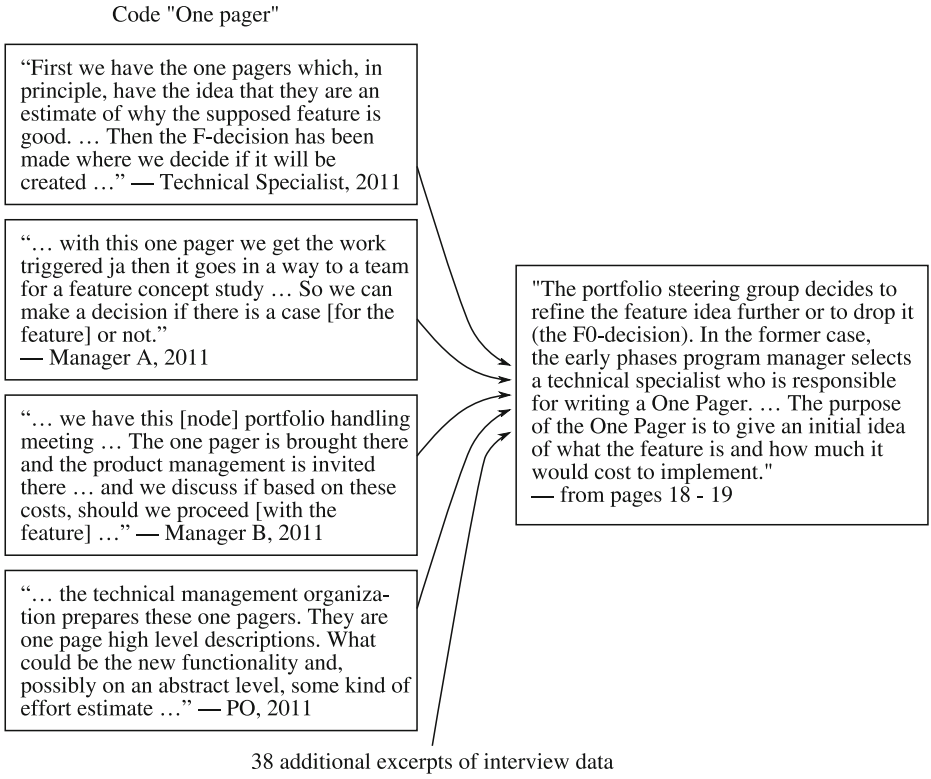


Fig. 2 An example of the data analysis process of the code "One pager" in the category "Planning artifact"

later on commercialized by Larman & Vodde as the Large Scale Scrum (LeSS) framework. Initially, a pilot Scrum team tried out the approach. Soon after, a few more teams were created and in quick succession the rest of the developers formed Scrum teams. The transformation did not stop there and the way of working was continuously improved, as reflected by the interviewees who called the transformation a "journey". The individual development teams had broad authority to change their team-specific operational management process. By 2013, many teams had made changes to the by-the-book Scrum process (Schwaber and Beedle 2002) to better suit their way of working. Details of the operational level requirements management process can be found in Section 5.4. A detailed analysis of the operational level requirements management process and its discrepancies with the prescribed Scrum process can be found in our previous publication (see Heikkilä et al. 2013a).

Before the transformation, the development had been arranged as a traditional plan-driven project organization. The requirements management process had followed the waterfall model: The release planning began two years before the date of the next release when the scope of the release was decided by the product management. Technical specialists then created an implementation plan for the requirements and the plans were handed to the developers for implementation. When the implementation was ready, the software passed through multiple testing and verification stages, and was finally shipped as a part of generally available products and as software updates.

5 Findings

In this section, we first answer RQ1.1, *What are the organizational roles involved in the requirements flow?* by describing the case organization structure, and in particular the roles and their responsibilities in the requirements flow. Second, we answer RQ1.2, *What are the processes of the requirements flow?* by describing how requirements flow thorough three processes: The *release project management process* provides information for requirements elicitation and analysis, new requirements are created and elaborated in the *feature development process* and the requirements are further elaborated and implemented in the *implementation management process*. The release project management process connects the release project to the strategy of the organization. Feature development is a continuous process that runs parallel to the release project management process. Due to the links between the two processes, the descriptions of the processes are somewhat interleaved. The implementation management process varies slightly between the different development teams and features, and thus the description should be considered an abstraction of the most common case. Third, we answer RQ2, *What are the perceived benefits of the requirements management processes?* by describing the requirements management process related benefits, and fourth, we answer RQ3, *What are the perceived problems related to the requirements management processes?* by describing the problems.

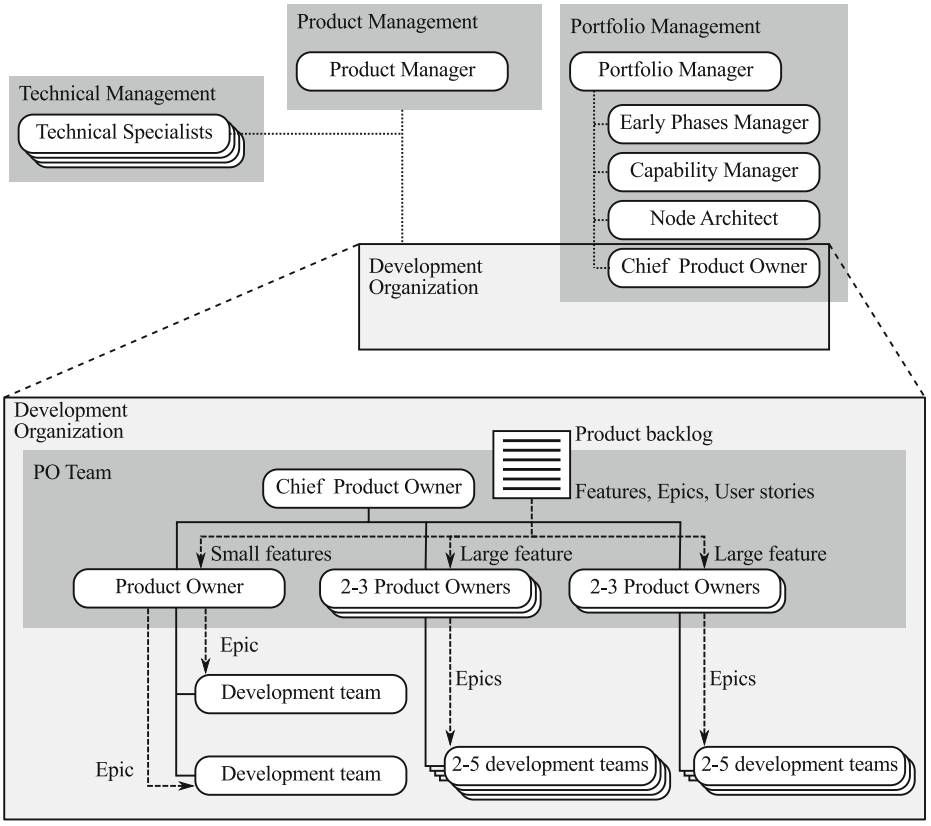


Fig. 3 The case organization

5.1 Case Organization Structure

In this section, we describe the roles and responsibilities of the members of the case organization in the requirements flow. We begin by introducing the hierarchical requirements model employed by the organization. Following that, we describe the organization of the product owners and the development teams. Then, we describe the organization of the other stakeholders. The organization of the stakeholders is illustrated in Fig. 3. Table 4 summarizes the stakeholders' roles and their responsibilities in the requirements flow. In addition to the formal organizational structure and communication channels, the case organization utilizes less formal communities of practice for knowledge sharing, coordination, technical work and organizational development. For more information on the communities of practice in the case organization, please see the case study by Paasivaara and Lassenius (2014).

5.1.1 Requirements Model

The case organization employs a hierarchical requirements model that has three layers of requirements. The requirements model is illustrated in Fig. 4. Requirements on all layers are stored in an electronic backlog management tool. *Features* are the highest level requirements. They have value to the customers independent of the other features. The implementation effort of a feature varies from a couple of months by a single team to implement to a year for ten teams to develop. *Epics* are split from the features. They are large, semi-independent functional requirements that create value to the customers on their own.

Table 4 Roles and responsibilities

Role	Responsibilities
Product manager	Long term product planning; Steering the feature development front end (as a member of the portfolio steering group); Steering the feature development (as a member of the development steering group); Release project planning (as a member of the product council)
Portfolio management	Refining long-term product plans; Managing the feature development front end (as a member of the portfolio steering group); Release project planning (as a member of the product council)
Chief Product Owner	Leading the PO team; Product backlog management and prioritization; Creating and prioritizing epics (as a member of the PO team); Managing and synchronizing the work of the development teams (as a member of the PO team); Steering the feature development front end (as a member of the portfolio steering group); Steering the feature development (as a member of the development steering group)
Product Owner	Managing and synchronizing the work of development teams (as a member of the PO team); Guiding development teams; Creating and prioritizing epics (as a member of the PO team); Creating, splitting and estimating user stories (with the developers)
Technical specialist	Supporting development teams; Managing the feature development front end (as a member of the portfolio steering group); Steering the feature development (as a member of the development steering group); Writing one pagers; Writing feature concept studies (with the developers)
Developer	Software development; Writing feature concept studies (with the technical specialists); Creating, splitting and estimating user stories (with the product owners), Unit testing

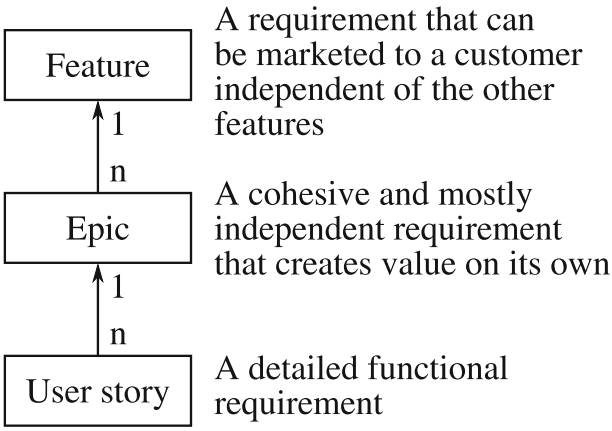


Fig. 4 The requirements model employed by the case organization

User stories are split from epics. They are the most detailed functional requirements and are mostly used by the development teams.

5.1.2 Product Owner Team

The organization of product owners deviates from the basic Scrum model (Schwaber and Beedle 2002). Instead of having team-specific product owners, a product owner team (PO team) was created to accommodate the large number of teams and requirements and the globally distributed structure of the organization. The PO team consists of a chief product owner (Chief PO) and ten product owners (POs). In order to mitigate personnel risks, the whole PO team is jointly responsible for the requirements and for managing and synchronizing the work of the development teams.

> If we want to know . . . how our development is progressing with the different backlog items, it is the PO team that pulls them together. . . . when we have several teams that are working towards the same functionality, they [the PO team] provide the big picture of how we are progressing – A Manager, 2011

The Chief PO is responsible for leading the PO Team. He acts as an arbiter between the development organization and the stakeholders external to the development organization. The PO team is responsible for managing the product backlog, which contains the requirements.

The POs rotate between teams when features are completed. Based on the size of the features, one PO might work with two cross-functional teams when both teams are developing their own small features, or a group of two to three POs might take the collective responsibility for one large feature developed by several teams.

5.1.3 Development Teams

The developers are organized in Scrum-inspired (Schwaber and Beedle 2002) teams of 6-7 persons. During the transformation, the previous function-based organization was dismantled and the members of the functional areas were assigned to the cross-functional teams. The team compositions were relatively permanent, although natural personnel change did

occur due to people leaving the organization, and due to layouts and recruitments. The intention was that the members of each team would gradually broaden their knowledge and eventually any cross-functional team could be assigned to develop any requirement. However, the managers soon realized that this goal is very challenging to achieve with a large, over ten-year-old product, since many areas of the product required very specific technical knowledge. Thus, the development teams, in practice, are usually assigned to work on requirements that best suit their competencies and previous experience.

> If the starting point is an organization with long history, naturally you have many kinds of skills and skill profiles in there, it is not irrelevant how you form the teams. Having an organization with 10, 20, maybe even more teams that are equally super skilled, it is a kind of utopia. – A Manager, 2011

5.1.4 Product Management

A product management function is responsible for the long term planning of the product from the business perspective, and crucial in aligning new requirements to the product strategy. A single product manager (PM) is responsible for the software part of the product and she is also the main point of contact between the development organization and the product management. The main communication channel between the development organization and the PM is the Chief PO. The other members of the development organization may contact the PM directly, if needed. The product manager communicates the product roadmap, which shows the long term plans for new requirements, to the development organization.

> ... the product management ... their job includes going around the clients and sales and marketing organizations so they can get input on what the customer wants and needs ... Furthermore, they create the roadmap which in practice shows for a couple of years what features are coming in what releases. – A Manager, 2011

5.1.5 Technical Management

Technical specialists support the product management in technical aspects of their product plans and have an important role in the front end of the requirements flow (see Section 5.2). The technical specialists also support the development teams and help them to plan and understand the technical aspects of the requirements. The technical specialists are people with extensive knowledge of telecommunication technology and each has a specific area of expertise, for example security or user interfaces.

> ... our node architecture is divided into functions that each have a responsible [technical specialist]. ... we have seen that it is necessary to hold on to these certain roles to keep the product sound. – A Manager, 2011

5.1.6 Portfolio Management

A portfolio management function refines the strategic, long-term plans created by the product management into concrete requirements development plans for the development organization. The portfolio management is led by a portfolio manager and it has the following members: An early phases program manager, a capability manager, a node architect and the chief product owner who is a member of the portfolio management function in addition to belonging to the product owner team.

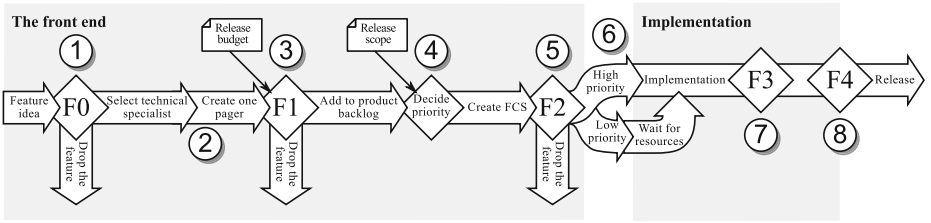


Fig. 5 The feature development process

> ...portfolio management ...is really an interface between the product management and R&D. [The portfolio management] consolidates what features are needed and what they cost and how much we can do and so forth. – A Manager, 2011

5.2 Feature Development Process

The feature development is a continuous, iterative and incremental process. The whole process is illustrated in Fig. 5. During the process the feature decisions are made in five feature-decision (F-decision) points numbered from F0 to F4. The front end of the feature development process spans from the feature idea to the F2-decision. The front end is called *the early phases* in the case organization. During the front end, the feature idea is refined and evaluated. The decisions F0, F1 and F2 are made by the *portfolio steering group* which consists of the product manager, the chief product owner, the portfolio management, the technical management, a release verification organization representative and representatives from various integration testing organizations. The steering group meets once a week and decisions that are related to any number of features can be made in a single meeting.

After the front end, during the feature implementation, the feature decisions F3 and F4 are made by *the development steering group*. It consist of the chief product owner, the product manager and testing and integration function representatives. Other stakeholders and product owners can participate when deemed useful. The planning artifacts created and used in the feature development planning process are summarized in Table 5. In the rest of this section we describe the process in detail. The numbers in parentheses refer to the process steps shown in Fig. 5.

Table 5 Planning artefacts

Artifact	Creator(s)	Contents
One Pager	The early phases program manager	Purpose of the feature
	A technical specialist	Tentative release target
		Technical impact
		Rough cost estimate
Feature Concept Study	A technical specialist	Cost estimate
	A product owner	Estimate of the required resources
	A virtual team of developers	Rough implementation and testing model
		Tentative implementation schedule

5.2.1 The Front End

The front end cycle is illustrated in Fig. 6. The whole process begins when the early phases program manager decides to propose a new feature idea **(1)**. The portfolio steering group decides to refine the feature idea further or to drop it (the F0-decision). In the former case, the early phases program manager selects a technical specialist who is responsible for writing a *One Pager* **(2)**. The length of the One Pager is limited to a single presentation slide and the maximum time given to writing it is two weeks. The purpose of the One Pager is to give an initial idea of what the feature is and how much it would cost to implement.

> ... [after] the One Pager request it is approximately two weeks when it should be ready. ... we have a portfolio meeting where we go thorough the One Pagers and make the F1-decision, [to decide if] will we process this [feature]. – A Manager, 2011

When the One Pager is ready, it is presented in a portfolio steering group meeting **(3)**. The steering group decides to either take it into further refinement or to drop it (the F1-decision). In the former case, the feature is added to the product backlog. The chief product owner then prioritizes the feature against the other features in the product backlog **(4)**. If he decides that the priority is low and there are no available resources, the feature has to wait until resources become available. If the feature is of high priority or when resources become available, the chief product owner selects a development team or teams and a product owner who are responsible for creating a *Feature Concept Study (FCS)*. A virtual team consisting of members from the team(s) and the product owner is formed. They are supported by a

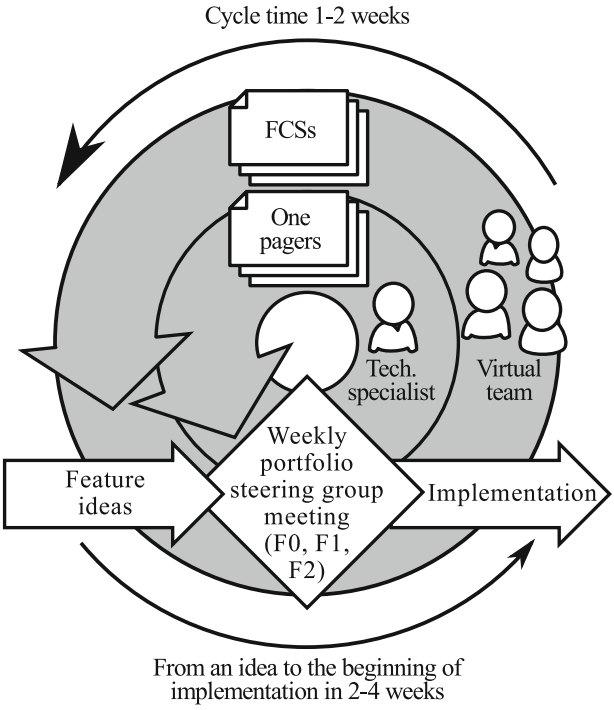


Fig. 6 The feature development front end cycle

technical specialist. The virtual team begins to write the FCS, which depicts information required to decide to develop the feature or not.

> ... [FCS] contains requirements, high level implementation model, feasibility, testing stuff. Checking these kinds of things. And naturally the costs. So we can come to this F2-point, where we see if we can implement it ... – A Manager, 2011

When the Feature Concept Study is ready, it is presented in a portfolio steering group meeting **(5)**. Based on the information in the FCS, the steering group decides to invest in the development of the feature or to drop it (the F2-decision). In the former case, the development of the feature can begin. If the Chief PO decides that the feature has high priority and development resources are available, the development begins immediately. Otherwise, the feature has to wait until development resources become available **(6)**.

> ... F2 is a permission to begin [the implementation], but it does not necessarily mean we begin. ... It [the feature] might be so small that we will not do anything with it for half a year. Let it wait on a shelf. Because we have seen already here that it is so small we can fit it in at some point nearer the end [the release]. – A Manager, 2011

5.2.2 Implementation

During the implementation, *epics* are split from the feature by the product owner team. The product owner team together with the developers, technical specialists and, often, the product manager estimate the epics and prioritize them into two categories which are the minimum scope and the full scope. Minimum scope contains epics that are mandatory for the feature to be publishable, and the full scope contains epics that are valuable but not mandatory. The set of epics in the minimum scope is called the minimum marketable feature (MMF) scope. The epics are then assigned to teams based on the technical competencies in the teams. To avoid integration problems and to reduce coordination effort, the epics are typically team specific.

> ... we are talking about this minimum marketable feature, MMF, ... what is really important and there we often talk on the epic level but also about individual user stories. ... Often we have product management there to instruct what is important and what is not that critical at the beginning. – A Manager, 2011

The Feature Concept Study and the product owners primarily guide the implementation, but the teams are also supported by other stakeholders, when needed. When the feature is relatively large, the development is initiated by several teams. More teams are added and more epics are created as the development progresses. When the feature is nearly completed, the number of teams is reduced, leaving a few teams to finalize the feature. When the chief product owner considers that a feature has progressed enough, he proposes an F3-decision in a development steering group meeting **(7)**. Based on the progress information on the feature, the chief product owner proposes a date when he believes the feature will be completed. This also means that the chief product owner can commit the feature to a release. The main goal is to communicate the feature development progress to the product manager. If the feature does not pass the F3-decision, the feature requires further development.

> ... we have F3 ... when we are quite far in the development we can, from the development side, say that we are going to get this feature done by a certain time so it will make it into the next release. – A Manager, 2011

After the feature has been implemented, tested, integrated, verified and documented, the chief product owner proposes an F4-decision in a development steering group meeting **(8)**. Based on the progress information, the steering group can then decide that the feature is ready to be included in a release. Otherwise the feature requires more finalization work.

5.3 Release Project Management Process

The release project management process is completed once for each version of the software. Figure 7 illustrates the process. The capital letters in the figure are referred to in the explanation below. Two simultaneous releases are usually under development at the same time. One release is more focused on new functionality and the other is more focused on maintenance, but both contain new functionality and updates. In total, two update versions and two maintenance versions are released every year. The organization is capable of doing more frequent software releases, but due to the high price associated with the localization and configuration of the system for the customers, some customers do not want to update their systems more often than every two or three years. Subsequently, the current release schedule is considered appropriate.

The planning of a new version release officially begins with a meeting of the *product council* **(A)**, which consists of product managers and portfolio managers. The meeting is held approximately two years before the date of the release. The key inputs to this meeting are the financial information and the product roadmap **(B)**. Based on the inputs, and on the information on the previous and ongoing release projects, the product council decides the tentative *release budget* and *release scope* for the release **(C)**. The release scope also inspires *feature ideas* for the release.

> . . . [The product council considers] what we can create by the release date and how much it costs. On the other hand our capability and the costs. . . . with the product management we make sure that we produce the input that they [the product management] need. – A Manager, 2011

Four to six months before the release date the release project management process enters the next stage. The product council decides what features should be included in the next release **(D)**. This decision is based on what features have passed the F3-decision **(E)**. Features that have not yet passed the F3-decision can be also included if the development progress information implies that the features will be completed in time for the release. Teams from less important features can be moved to more important features that are behind the schedule, but this is rarely necessary. After this stage, the release can be made public and the marketing of the new features can begin **(F)**.

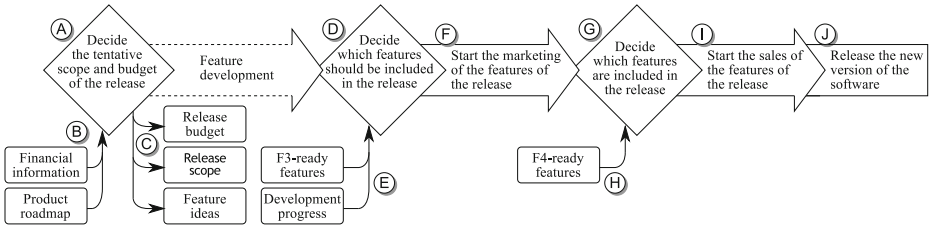


Fig. 7 The release project management process

> ... after the decision [(D)], we can start marketing the release ... then we can include in the release features that have reached the F3-status ... in practice, we have not always reached [the F3-status] but we have a risk level ... that we will mostly make it [ready in time for the release]. – A Manager, 2011

Two to three months before the release date the product council decides which features will be included in the release **(G)**. All included features must have passed the F4-decision **(H)**. After this stage, the included features can be sold and binding contracts regarding the release can be made **(I)**. The final step in the release project management process is the release of the software **(J)**. After this step, the responsibility for the released features is officially transferred from the node development organization to the support organization.

5.4 Implementation Management Process

The development teams utilize a Scrum-inspired implementation management process. Development is paced by two-week long development iterations (a.k.a. sprints). Figure 8 illustrates the schedule of the development sprints. The Roman numbers in the figure refer to the process description steps that are explained below. The first step of the process is the assignment of epics from the product backlog to the development teams **(I)**.

> In a way we have Scrum, we have epics and then epics are split into smaller user stories. When they are approaching the time of implementation, they are split and elaborated and possibly split [further]. – A Developer, 2013

At the beginning of the development of a new feature, the product owner(s) conduct a backlog grooming session **(II)** with the team(s) working on the feature. The purpose of this grooming session is to create user stories based on the epics and to coordinate the development of the epics between the teams. During the development of a feature, each development team has a backlog grooming session on every other Wednesday. In the grooming sessions, the product owner(s) and the team members elaborate, estimate and prioritize user stories. The goal is to have at least two sprints worth of work for each team in their team backlog at all times.

> ... in halfway into the sprint ... we estimate upcoming work and possibly we groom the upcoming user stories, so if we have some epic which needs to be implemented

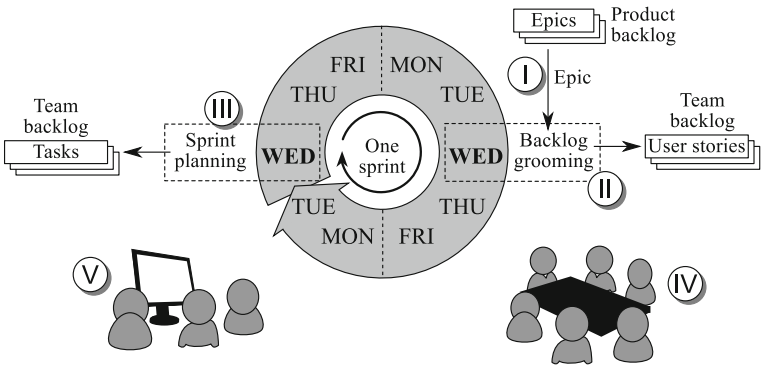


Fig. 8 The implementation management process

> we split it . . . we try to create user stories that are sensible, that we roughly know what they include. – A Developer, 2013

Sprint planning **(III)** is conducted on every other Wednesday alternating with the backlog grooming sessions. Sprint planning is conducted in two parts. The first part is akin to a development status meeting of the teams assigned to the feature. Each team conducts the second part individually. In the second part, the teams split their user stories into concrete development tasks which are then added to the team's backlog.

> . . . [the first part] is like a traditional project meeting, everyone goes there and listens what is coming next. [In the] second part . . . we split [the work] and so on . . . – A Developer, 2013

The teams' work is not strictly tied to the sprint cycle. The goal is to minimize the lead time by having as few user stories simultaneously in progress as is practical. Additional user stories can be added at any time if there is not enough work for every member of the team **(IV)**.

User stories are typically demonstrated to the product owner and closed as soon as they are ready **(V)**. This enables the development of any dependent user stories to begin immediately and makes the completed user story a part of "the legacy",[4] which means that the team is no longer responsible for solving integration issues that affect the user story.

5.5 Benefits

5.5.1 Reduced Development Lead Time

In the previous model, the release planning was conducted during the first six months of each approximately two-year release project. The changes that could be made to the release after the first six months were typically very small. If a requirement missed the release planning window, the lead time from a customer request to a public release of a new requirement could be over three years. After the transformation, the amount of planning and research before requirement implementation was drastically reduced, which also contributed to reducing the lead time. Changes to the requirements (features, epics and user stories) could be made flexibly by the corresponding decision makers.

> . . . [previously] releasing one package took 18-24 months. In the beginning we performed this system planning which took maybe half a year. And if you did not get the right contents in the release during the first half a year . . . it was immensely difficult to get any changes into the project. . . . If some essential functionality was missing from it, we missed the train, I had to wait to the end of the current project and then the two years after [that]. Which was a very long time. – Product Manager, 2011

5.5.2 Increased Flexibility

The new process was seen as an improvement to the flexibility of development. The new release management process allowed changes in the contents of a release relatively near the release date. The feature development schedule was no more tied to the release schedule,

[4] A term employed by the interviewees.

which immensely decreased the lead time of the feature development. The portfolio management could control the release schedule of requirements by adjusting the minimum marketable scope and by re-assigning development resources.

> Now it is like, okay, let's add it to the list. And no worries about where we are going with the change. It's there and in a way nothing was changed even though a new thing was added to the list. I think it is a really good improvement. The flexibility is on another level. – Product Manager, 2011

5.5.3 Improved Planning Process Efficiency

During the F0-F2 steps in the feature development process, the sunk costs were relatively small and the early identification of unprofitable or otherwise infeasible features saved development resources. In addition, by employing the minimum marketable feature concept, the case organization was able to concentrate on developing the most important parts of the features.

> What is good in it [the F-decision process] is that ... it in a way divides the decision making, which is a good thing. We can cut it [the feature] at any point, ... if we see that the feature passes the time window or otherwise. It gives structure to the decision making and enables us to make smaller decisions and in that way separate the feature decision from the release decision. – Manager, 2011

5.5.4 Increased Developer Motivation

Three aspects of the new development and requirements management processes contributed to the increased motivation of the developers. First, the development teams were empowered to change and improve their team-internal work practices and to prioritize their work on the task level. Second, the developers had more opportunities to broaden their areas of technical competence than in the previous model. Third, the developers were included in the feature planning already at the front end, which allowed them to contribute to the planning and gave them visibility to the purpose of the feature.

> ... the gang is excited that they get to try out something new. That is, at least as long as the whole user story is not in a completely new area, that there is just some small part, for example, from some new area to work on. ... – Product Owner, 2011

> ... one person from the product management involved in [the feature] traveled from here to [Hungary] and had a one-day workshop. Why [the feature] is needed for the customer, what information they get, what kind of reasoning [is] behind this feature. ... It was a motivation boost for the team, to see that what they are doing ... means something for [the customers]. – Developer, 2011

5.5.5 Direct and Efficient Communication

In the previous, plan-driven process the requirements related communication was hierarchical and mostly based on e-mails and documents. In the new requirement management process, the members of the development organization communicate directly with their peers and other stakeholders. Face-to-face communication was considered to be a much more efficient way to communicate requirements information between the different stakeholders than the previous way.

> You can ... just walk in and ask that what is this thing. Compared to [the previous situation] where our tester sat in a department and we, the design, sat here. The distance to there was so long that you never walked there. Communication was difficult because it was always based on an e-mail or something and it didn't work.
> – Product Owner, 2011

5.6 Problems

5.6.1 Overcommitment Caused by Pressure from the Product Management

According to the interviews, the product management still worked in the "old world" way. They requested long-term requirements development plans from the PO team, which were not available in the new release management process, and pressured them to give premature commitments when the release date was approaching. This caused overcommitment by the development organization, which left very little leeway in the development schedule and decreased the flexibility of the development.

> ... perhaps the product management is not in the new way of working, it easily goes with the old model that we plan one big release ... it feels like we plan a big future release and see what can fit in it. And then what happens along the way is that all the time new things are coming in which don't fit in the scope. And then we need to remove something. It perhaps leads to a spiral where we feel that we are late all the time and we must leave something undone because we don't leave any buffer any more.
> – Product Owner, 2011

The case organization tried to mitigate this issue in two ways. First, they tried to improve the predictability of the development by increasing the detail level of FCSs and by increasing the amount of slack in effort estimates. Second, they had created the concept of a minimum marketable feature, which was the set of epics they could commit to delivering by the next release on a high probability level.

5.6.2 Balancing Planning Effort

The case organization had difficulties balancing the effort spent on planning the requirements and the demands from the development teams for more comprehensive implementation guidance. After the transformation, the estimation accuracy was initially quite low, as the features were only estimated very superficially. However, this resulted in the notable underestimation of effort of some features and massive over-estimation of others.

The organization tried to mitigate this by creating detailed feature concept studies. Detailed FCS provided the teams with information on how the feature should be implemented, which also improved effort estimates. On the other hand, creating a detailed FCS took effort and time, which was against the original purpose of the whole front end process, and the stakeholders had difficulties understanding the feature when the FCS contained many implementation details. Identifying the right level of detail for feature concept studies was an ongoing issue in the case organization during the case study.

> ... we have had an overcommitment phenomenon. ... because of weak analysis. Because we pulled these estimates out of a hat. ... but now we have identified it and found a cure for the next round, we will make this Feature Concept Study before we start.
> – Product Owner, 2011

5.6.3 System-Level Planning, Documentation and Problem Solving

In the previous, plan-driven development model, the project managements' responsibilities were clearly defined and the role of a project office was very central. The project mangers and technical specialists of the project office had precise responsibilities in the projects. In the new model, the PO team assumed the responsibility for directing the development teams, but it was unclear who took care of the problem solving and other tasks the project office previously handled. Many of those tasks were originally out of the scope of the PO team's responsibilities, but had to be addressed somehow. The tasks included system planning guidance, non feature-specific problem reports, system documentation and external change requests. In order to address these tasks, the PO team had initiated an additional bi-weekly meeting. In addition, the managers were planning to start organizing communities of practice (Paasivaara and Lassenius 2014) around the system-level topics in order to mitigate the problem.

> ... We do not have anything else than these cross-functional teams and this [the documentation] is not related to any feature ... things are falling between the chairs all the time. It is a problem. ... This allows us to see things that must be done, of those that we did previously. Most of them were mandatory, it seems. ... they come as bit of a surprise, that hasn't anyone handled even this thing. – Product Owner, 2011

5.6.4 Defining the Responsibilities of Product Owners

The product owners came from varying backgrounds. Some had been in a more technical role before the transformation while others had had a more business-oriented role. After the transformation, the POs had, in theory, a very business-oriented role. Nevertheless, the development teams expected their PO(s) to help them also in the technical implementation planning but some POs did not have sufficient technical knowledge to be of assistance. This was considered a challenge by the developers and the POs, but the Chief PO was of the opinion that the different backgrounds of the POs made the PO team more capable overall. At the end of our case study period, the case organization was still trying to overcome this challenge.

> Another thing that is challenging is that, in this Scrum world, the Product Owner should not need to be a very technically skilled person, in principle. But with us, in practice, they must have [a] very good technical background. Otherwise it doesn't work with the way we are doing things now. ... Because they are the only persons who have the big picture. If that person does not have technical knowledge he cannot see what kind of problems might be coming. And even defining and splitting user stories will be difficult if you do not know enough about the area. ... And the other thing is communication, if that guy does not understand the questions that the team is discussing then he cannot help. – Developer, 2011

5.6.5 Growing Technical Debt

There were signs that the technical debt in the system was growing after the agile transformation. The developers were focused on quickly producing the individual user stories during the two-week sprints and less emphasis was given to the overall internal quality of the system. The development teams also lacked the skills to do long-term technical and architectural planning. Before the agile transformation, the technical specialists did the

technical work that was not requirement-specific and performed long-term technical planning. Following the transformation, the technical management function was considerably cut down and the remaining technical specialists did not have time to perform these tasks. Initially, technical specialists and architects were considered redundant in the new organization, but the managers soon realized that the roles were required in order to guide the overall development of the system and to upkeep the consistency of the system architecture. As the developers become more capable of doing requirement-specific technical planning, the technical specialists focused more on the consistency of the system architecture.

> We don't want it to occur that when our features are developed our cross-functional teams create them quickly, and then we notice two years later that our architecture is such a mess that adding anything to it is very expensive. Using these specific roles [technical specialists and architects] we strive to maintain the consistency of the architecture because this [development] organization won't do it any more. – Portfolio Manager, 2011

5.6.6 Balancing Between Development Efficiency and Building Generalist Teams

Initially in the agile transformation, the goal of the development organization was to create cross-functional generalist teams that could implement requirements in all components of the software. However, the managers quickly realized that many components were technically very difficult and required years of experience to completely understand. This had, in several occasions, caused very long lead times (up to one year) before a team could implement anything useful in a component in which they had no preexisting experience in. It had also resulted in features that were not technically sound. Balancing between the development efficiency and building generalist teams was seen as difficult especially near the release date when the pressure to get features completed was mounting. The portfolio steering group had started mostly assigning epics to the teams that had preexisting competency in the affected components.

> . . . building the competencies has been one of the biggest challenges. . . . we have very difficult products where the transfer [of knowledge] is very challenging, it cannot be done in a couple of sprints, it requires several months, in practice. We've had to yield in that, we had to give it to the best [team] . . . – Scrum Master, 2011

> For example in this [feature] we have observed that if the POs allocate these user stories randomly to the five teams it leads into a situation where the initiation cost, that you learn the thing and the part you should change, it becomes excessively hefty and costly. And following that we do increasingly so that the teams . . . concentrate on the epic level on certain things. – Portfolio Manager, 2011

5.6.7 Insufficient Understanding about the Development Team Autonomy

Many developers in the case organization expected that their product owners or Scrum master would provide detailed directions on what they should work on. The developers also expected that they would be given detailed requirements specifications. However, in the new, agile organization they were expected to find out the information themselves or ask for help. Initially, many developers were not interested in improving their work practices and tools.

On the other hand, the product owners tended to prioritize new features over system improvement and the developers did not have time to do improvement work in addition to

implementing the features. The development organization attempted to mitigate this issue by making each team take in at least one system improvement user story every sprint. These issues started to alleviate when the development teams and product owners got more experience in the agile way of working.

> ... if everything that must be completed comes from the Product Owner's backlog, the people [developers] are not in a good place. ... If I am in a team and I think creating test automation is the most important thing in the world because it eases my work so much [that] it is worth doing now, then if the only way to get it done is to get it into the backlog where it is prioritized wearing a business manager hat, probably to a very low priority level, then the team empowerment pretty much fails. – Developer, 2011

6 Discussion

In this section, we first discuss the answers to our research questions and compare them with the previous research. Following that, we discuss the contributions we make to practitioners. Finally, we discuss the limitations and the threats to the validity of our research.

6.1 RQ1: How is the Requirements Flow from the Strategy to a Release Managed?

The case organization employed a multi-level requirements management process in which each level had a different planning horizon, purpose and decision makers. On each level, the decision makers with the most knowledge on that planning level made the decisions. The release project management process put the long-term strategy of the company into operation by deciding the tentative scope of the next release and by identifying ideas for potential new requirements. The feature development process was a continuous process in which the requirements were prioritized, resourced and scheduled. The implementation management process was inspired by the single-team Scrum process, but the Scrum rules were not strictly followed. In a previous study (Heikkilä et al. 2013a), we analyzed the implementation management process quantitatively. Please see that study for a detailed discussion on the reasons the implementation management process did not strictly follow Scrum. Figure 9 illustrates the three management processes and the main information flows between them.

The interfaces between the different levels of management were implemented by roles that had responsibilities on adjacent levels. The product council was responsible for implementing the product development roadmap. The portfolio management was the interface between the release project management and the feature development process. The chief product owner, who belonged to the portfolio management, conveyed the requirements decisions and feature information to the product owners in the PO team and the product owners conveyed the information to the development teams. In addition, the technical specialists conveyed technical information on the feature development and implementation levels. Requirements implementation progress information flowed from the development teams to the feature development process, and from the feature development process to the release project management process.

According to Vähäniitty (2012), several authors have proposed that a hierarchical model for requirements is necessary in a large-scale agile development organization. The hierarchical model has been proposed to provide several benefits over a simple, list-based product backlog (Vähäniitty 2012; Leffingwell 2011). Hierarchical requirements can be split into

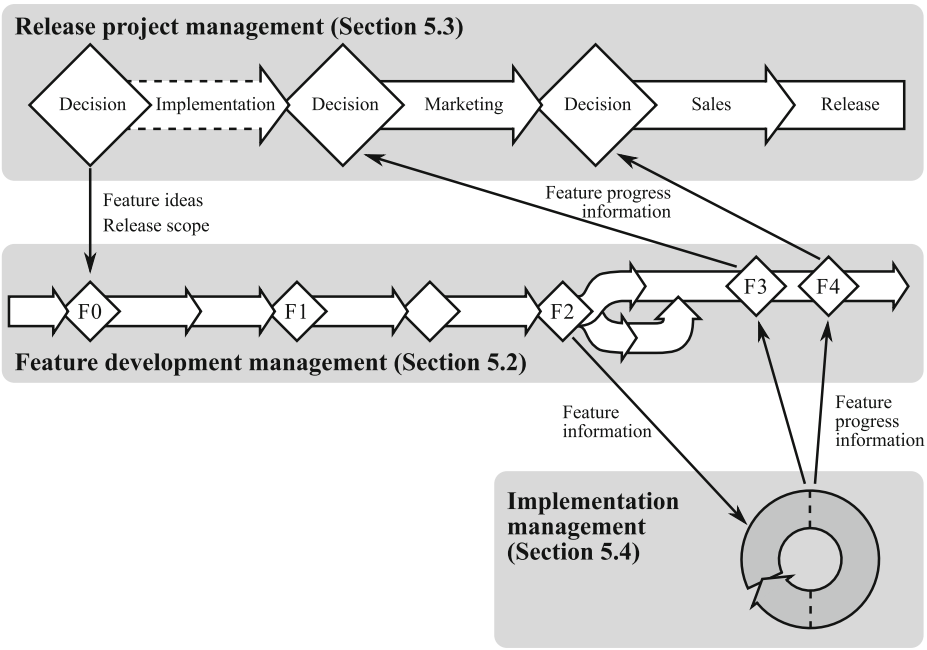


Fig. 9 The management processes and main information flows

smaller parts that can be prioritized against each other, which allows the development to focus on the most important parts of each higher-level requirement. A hierarchical backlog is more manageable than a long flat list of requirements that are of different sizes and have been elaborated to different levels. Managers can more easily prioritize and elaborate high abstraction level requirements, while detailed, low abstraction level requirements are understood more easily by the developers. Although the hierarchical requirements model seemed to be well suited for managing the functional requirements, the management of the system-planning, documentation and problem solving was a problem, as such work could not be associated with a single requirement. Our case study supports the previous findings that hierarchical requirements models are applicable and beneficial in large-scale agile development organizations. However, we also find that special attention must be paid to the management of work that is not requirement specific.

The hierarchical requirements model used by the case organization was similar to the one proposed by Leffingwell (2011). However, the framework proposed by Leffingwell (2011) is based on nested layers of iterative and incremental planning. In his framework, release project planning is performed every 2-3 months and the goal is to tentatively plan the contents of the next *potentially shippable increment (PSI)*. Each PSI should be of shippable quality, but the actual shipping date of the software is based on a product roadmap. Unlike in our case organization, features or epics are not assigned to specific teams. The splitting and assignment of epics and features are done in release project planning events where the whole development organization gathers to plan the next PSI. Compared with Leffingwell's framework, the management model in our case organization was asynchronous and flexible. The feature development was decoupled from the release schedule although the release plans affected the feature priorities. The decoupling allowed the feature development

process decision makers to be more flexible in the scheduling, prioritization and resourcing of features. Our findings indicate that this method of requirements management and planning was considered quite suitable for the organizational and technical context of the case organization.

The organization of the development teams and product owners was similar to the one proposed by Larman and Vodde (2010) but the organizational structure in the case organization was less rigid than theirs. Instead of a strict division into technical product areas and feature teams, the case organization had more flexible organization consisting of feature specific groups of teams and a product owner team. However, due to the challenges the teams had with the learning overhead, the portfolio steering group had begun to assign teams to features on technical areas they had previous knowledge of. This made the de facto organization of development teams somewhat similar to the one proposed by Larman and Vodde (2010). Our findings suggest that the division of development teams into technical areas may be beneficial to avoid excessive learning overhead when the system is large and technically complex.

In rare cases, additional teams were assigned to develop features that were not progressing as expected. According to Brooks' Law, "Adding manpower to a late software project makes the project later" (Brooks Jr. 1975). As Brooks himself states, the law is an outrageous oversimplification. Brooks' Law considers a scenario where new developers are added to an existing project team, the development process follows the waterfall model and the development tasks are highly dependent and require intricate coordination between the old and new developers. The epics in the case organization's requirements hierarchy were semi-independent and mostly team-specific, which reduced the need for inter-team communication. Instead of assigning individuals to work on specific projects, teams were assigned to work on features. Most of the time, teams were assigned to work on features that did not require excessive amount of learning. A reasonable amount of learning effort was considered an acceptable cost of expanding the teams' knowledge. The team compositions were not changed in order to rush features. Furthermore, the organization did not follow the waterfall model. These aspects might explain why Brooks' Law was not seen as a problem in the case organization regardless of the overscoping that was still a problem.

Inayat et al. (2015) identified the following 17 requirements engineering practices that were adopted in agile software development: 1. Face-to-face communication, 2. customer involvement, 3. user stories, 4. iterative requirements, 5. requirements prioritization, 6. change management, 7. cross-functional teams, 8. prototyping, 9. testing before coding, 10. requirements modeling, 11. requirements management, 12. review meetings and acceptance tests, 13. code refactoring, 14. shared conceptualization, 15. pairing for requirements analysis, 16. retrospectives and 17. continuous planning. We identified many of these practices in the case organization, but most were adapted to the large scale of the organization. The developers did not typically communicate requirements knowledge face-to-face with the customers (1, 2). Instead, the product owners and the product manager represented the customers. The development teams managed requirements as user stories (3), but requirements were also managed on higher abstraction levels in the form of epics and features. Instead of the simple iterative elaboration of user stories (4), the features were iteratively elaborated in the development front end and after they were split into epics and user stories. Requirements prioritization was performed on the user story level (5), but also on the feature and epic levels. Change management (6) was based on the flexibility of the feature development process. The case organization strove to create cross-functional teams (7), but only in a specific technical area. The requirements were managed with a product backlog (11), but instead of using only index cards or user stories, the case organization employed a three-level requirements

model. Instead of pairing for requirements analysis (15), the requirements were analyzed by many different stakeholders, including product owners, developers and the product manager. Continuous planning (17) was performed in the feature development process on the feature level and in the implementation management process on the epic and user story levels.

6.2 RQ2: What are the Perceived Benefits of the Requirements Management Processes?

The new way of working created most of the benefits the case organization was expecting. The benefits are summarized and compared to previous work in Table 6. The main goals the organization had were to improve the flexibility of the development organization and to improve the motivation of the developers.

Detailed predevelopment requirements studies had been replaced by the stepwise elaboration of requirement information in the new feature development process. This had drastically *reduced the lead time* from a minimum of two years to a minimum of a few months. The new requirements management process *increased the flexibility* of the feature development. This allowed the product council to flexibly react to changes in the market and change the emphasis of the feature development. This was viewed as a major improvement especially by the product management. Furthermore, the recent rise of software-defined telecommunications networks is expected to further emphasize the competitive advantage created by short development lead times (Batista et al. 2015). Korhonen (2013) also found that the flexibility of the organization improved during the agile transformation of a large organization. In a survey study of a large organization, Laanti et al. (2011) identified the increased flexibility as one of the top benefits from agile development. Increased responsiveness to change has been claimed to be a benefit from agile RE in general (Heikkilä et al. 2015a; Inayat et al. 2015). Although the case study indicated that the organization considered the new requirements management process a success, we cannot say whether the

Table 6 Summary of the benefits from our study and related work

Benefit	Description	Related work
Reduced lead time	The time from a feature idea to feature release is much shorter.	Software-defined telecommunications networks emphasize the advantage of short lead times (Batista et al. 2015).
Increased flexibility	Changes to feature plans and priorities can be made efficiently.	Agile transformation increases flexibility (Korhonen 2013; Laanti et al. 2011). Agile RE improves responsiveness to change (Heikkilä et al. 2015a; Inayat et al. 2015).
Improved planning process efficiency	Infeasible or unprofitable features are dropped early.	Agile RE reduces process overheads (Heikkilä et al. 2015a; Inayat et al. 2015).
Improved developer motivation	Empowerment, learning and visibility increase developers' motivation.	Developers are more motivated in an agile than in a plan driven organization (Noordeloos et al. 2012; Korhonen 2013).
Improved communication effectiveness	Emphasis is on efficient face-to-face communication instead of documentation.	Agile RE reduces communication problems (Inayat et al. 2015).

selection of features was better than before the agile transformation, as such data was not available to us.

An additional benefit from the new feature development process was the *improvement of the planning efficiency*, since unprofitable or otherwise infeasible requirements could be identified and discarded before much effort was spent on studying and specifying them. The findings support the previous research on agile RE (Heikkilä et al. 2015a; Inayat et al. 2015) that claims that agile RE reduces process overheads. The de-emphasis of the predevelopment requirements specification and documentation also forced the development teams to improve their understanding about the requirements and the overall system, which created some learning overhead. However, the learning overhead was expected by the case organization as a cost of reduced lead time and increased flexibility. The case organization did not measure developers' use of time for learning. Subsequently, the quantitative analysis of the requirements planning efficiency was not possible.

One of the basic agile principles is the empowerment of the development teams to form their own team work practices and to collectively take responsibility for planning and managing their work (Moe et al. 2010). According to software engineering research (Verner et al. 2014; Beecham et al. 2008), many factors of software engineers' motivation are culture and context dependent. However, existing literature also suggests that there are factors that are cross-cultural and applicable to all knowledge-intensive work (Pink 2011). These factors can be summarized as autonomy, mastery and purpose. These factors may explain why the *developer's motivation was clearly improved* following the agile transformation and the new requirements management process. The teams were given the autonomy to change their work practices on the team level, they were given opportunities to learn and master new areas of the system and by including them in the front end of the feature development, they were given better visibility to the purpose of the upcoming requirements. Previous studies on agile transformations have identified the increased developer motivation as a benefit from the transformation from a plan-driven to agile organization (Noordeloos et al. 2012; Korhonen 2013). The findings of our case study support the previous findings, which suggests that increased developer motivation is a significant effect of a transformation from a plan-driven to an agile development process.

Different theories, such as the media richness theory (Daft and Lengel 1984) and the media synchronity theory (Dennis and Valacich 1999), have been proposed to explain the differences in communication efficiency when different tools and mediums are used. In general, such theories agree on that face-to-face communication is the most effective way to convey complex and contemporary information. These theories also explain why the change from the previous, document-driven requirements process to the new, agile process was perceived to *improve the effectiveness of communication* in the case organization. Although the increased reliance on face-to-face communication may introduce information bottlenecks, we did not identify such problem in the case organization. Inayat et al. (2015) found that agile methods, in general, reduce requirements communication problems due to the frequent face-to-face communication and iterative requirements elaboration, which also supports our findings.

6.3 RQ3: What are the Perceived Problems Related to the Requirements Management Processes?

The problems we identified in the case study are summarized and compared to previous work in Table 7. Several problems were caused by the previous, plan-driven way of working. Developers had had quite a narrow area of expertise and a limited knowledge of the overall

Table 7 Summary of the problems in our study and related work

Problem	Description	Related work
Balancing planning effort	Detailed planning takes too much effort, but stakeholders request detailed plans.	Requirements management and iterative planning are difficult in large-scale agile development (Laanti et al. 2011).
Overcommitment caused by the product management	Product management pressurizes the development to create unrealistic plans.	Agile RE reduces overscoping (Inayat et al. 2015; Heikkilä et al. 2015a).
Insufficient understanding of the development team autonomy	Both managers and developers have difficulties in understanding the additional responsibilities the teams have.	The role of middle management is a challenge in large-scale agile transformations. Team autonomy and empowerment are success factors in large-scale agile transformations. (Dikert et al. 2016)
Defining the role of product owners	High level of technical and business-oriented knowledge is required from the POs who rarely have both.	Split the PO role between a technical and a business person (Paasivaara et al. 2012a). Create a cross-functional product owner team that shares the responsibilities for technical and business guidance (Bass 2015).
Balancing between generalist and specialist teams	Learning overhead is too high in some technical areas, but specialization decreases flexibility.	Limited specialization is beneficial in large-scale agile development organizations (Leffingwell 2011; Augustine 2008; Larman and Vodde 2010; Moe et al. 2014).
System-level planning, documentation and problem solving	No official way to handle planning, documentation and problem solving that is not team specific.	Dependency coordination is challenging in large agile organizations (Wiklund et al. 2013; Laanti et al. 2011). PO teams may have to coordinate requirements management and planning (Bass 2015; Eckstein 2014).
Growing technical debt	Teams do not have the time or skills to do long term system planning.	Explicit system-level planning may be required to avoid growing technical debt in agile development (Korhonen 2013; Heikkilä et al. 2015a; Inayat et al. 2015). A special role might be needed for the management of the internal quality (Moe et al. 2014).

system, and the product management had employed detailed schedules and effort estimates. The case organization had difficulties in finding a good *balance in planning effort*. On the one hand, developers requested more detailed plans and the product management requested more detailed estimates, and on the other hand, the case organization wanted to keep the detail level of plans low in order to keep the feature development flexible and the lead time short. A similar problem was identified by Laanti et al. (2011) in a survey study of opinions towards agile development in a large organization. They found that the respondents perceived performing requirements management and iterative planning as the second biggest challenge, the deployment of agile methods being the biggest challenge.

In our case organization, especially when the date of an external release was approaching, the development organization made *overcommitments caused by the external pressure* from the product management, which led to unexpected scope changes late in the release project. Reduced overscoping has been proposed to be a benefit from agile requirements engineering due to the constant interaction with the customer (Inayat et al. 2015; Heikkilä et al. 2015a). In our case, the pressure to overscope came from outside of the development teams and could not be completely mitigated by the agile method employed by the teams.

Due to the pretransformation, plan-driven process where implementation plans were handed down to the developers, some development teams had *insufficient understanding about the development team autonomy*. They still expected that detailed implementation specifications were handed to them by the product owners. However, this was against the role the product owners had in the organization. Dikert et al. (2016) identified that the role of middle management is a challenge in large-scale agile transformations. They recognized that middle managers have to resist the tendency to command and control development teams. They also found that the team autonomy was an important success factor in agile transformations. Their findings support our findings on the importance of understanding the development team autonomy and empowering them to manage requirements on the development team level.

The developers' requests for detailed implementation guidance combined with the relative recent agile transformation created a problem with *defining the role of product owners*, since product owners, according to Scrum, have a very business-oriented role. They are not supposed to give developers detailed technical tasks and are not expected to be knowledgeable of the implementation details. Paasivaara et al. (2012a) describe how one large organization solved the problem by dividing the product owner role between two people. One had more technical skills and the other more business-oriented skills. Bass (2015) found that despite receiving support from a technical architect, development teams had to put lots of effort into understanding the product architecture. These results suggest that a technically skilled product owner role is beneficial even in mature large-scale agile development organizations despite the original, business-oriented purpose of the role in the Scrum literature. However, people with both skills are very rare and more often the issue is solved with a cross-functional product owner team or by splitting the product owner role between multiple people with different responsibilities (Bass 2015).

The rest of the problems were related to the need to adapt the agile ways of working to a large organization and a complex system that was developed. The case organization found that cross-functional teams capable of developing any feature in end-to-end fashion were infeasible and struggled to find a good *balance between development efficiency and building generalist teams*. Although the authors of early Scrum literature were quite determined that the cross-functional end-to-end team was the only option (Schwaber and Beedle 2002), later authors have accepted that there is a need for limited specialization on both the system and team level, especially in large development organizations (Leffingwell 2011; Augustine 2008; Larman and Vodde 2010). The solution proposed by Larman and Vodde (2010) is to divide the development organization into technical areas based on the structure of the system and allow specialization between those technical areas. This kind of arrangement was empirically studied by Moe et al. (2014) and found to be quite successful. Heikkilä et al. (2015b) studied a large, agile development organization where the teams were divided into two technical areas that mirrored the structure of the system. All teams performed release planning together in release planning events, which helped them to coordinate the work between the two technical areas.

In the agile single-team, single project sweet spot, the team (including the product owner) takes responsibility for all work, including the *system-level planning, documentation and problem solving*. In a large organization that is developing a complex system with a massive legacy, there are many cross-feature or feature-independent tasks that need to be coordinated. In our case organization, the development teams focused on developing features and they lacked knowledge of the system-level work. The product owner team sought to solve the problem by taking the lead in the coordination of the cross-feature or feature-independent work. Wiklund et al. (2013) and Laanti et al. (2011) also identified the coordination of dependencies and co-operation as challenges in large agile organizations. Previous research on coordination needs in large-scale agile development organizations has identified that product owner teams have coordination tasks regarding requirements gathering and prioritization, architectural planning and release planning (Bass 2015; Eckstein 2014). Noordeloos et al. (2012) identified the lack of decision and design documentation as a significant problem in an agile transformation, as reversing incorrect decisions was difficult without decision documentation. Our findings suggest that product owner teams might be required to also coordinate system-level documentation and problem solving in addition to managing the requirements.

Our findings and the previous research (Wiklund et al. 2013; Laanti et al. 2011) suggest that inter-team coordination and communication are major problems that a large organization adopting agile methods needs to solve. Our case study implies that these problems can be mitigated by establishing requirements oriented and technology-oriented coordination organs above the individual team level. In our case, these coordination organs were, respectively, the product owner team and the technical specialists. On a more general level, Scrum-of-Scrums (Schwaber 2007) and communities of practice (Larman and Vodde 2010) have been proposed as solutions for inter-team coordination in large agile organizations. The efficiency of Scrum-of-Scrums to coordinate work has been found to be insufficient beyond a certain size limit due to the large number of participants and the divergence of the topics that are covered (Paasivaara et al. 2012b). The case organization had also employed communities of practice to mitigate the problem and the results have been quite promising (Paasivaara and Lassenius 2014). Employing communities of practice is also better aligned with the principle of self-organization in comparison to the centralized coordination by a product owner team.

Our findings support the previous research (Korhonen 2013; Heikkilä et al. 2015a; Inayat et al. 2015) that suggests that explicit architectural and system-level planning may be required to avoid *growing technical debt* in agile development. The case organization tried to mitigate this problem by partially restoring the technical specialist positions that had been deemphasized in the initial agile transformation. The case organization also employed communities of practice to mitigate this problem (Paasivaara and Lassenius 2014). However, since communities of practice are self-organizing, their success is closely tied to the enthusiasm of the members of the organization. Moreover, if developers do not have knowledge of the growing technical debt, they cannot organize a community of practice to mitigate the issue. Thus, architect or technical specialist positions may be needed to raise awareness of the problem and to initiate communities of practice to solve it. Moe et al. (2014) described how an Ericsson subsystem development organization (that does not overlap with our case organization) in Sweden and China employed special Technical Area Responsible (TAR) role to manage the internal quality of the subsystem and for technical inter-team coordination. The most experienced developers were assigned the TAR role. Their results indicate that the TAR role was an effective way to manage internal quality and technical inter-team coordination, but it was quite burdensome for the individuals who had the

role. The TAR role was quite similar to the role the technical specialists had in our case. The main difference is that the TARs were developers that were given managerial responsibilities while, in our case, the technical specialists needed to work more closely with the developers.

6.4 Practical Implications

Based on the existing literature and our case study, we can put forward the following three initial implications for applying agile development methods and agile requirements management in large organizations: 1. Agile methods can work on the team-level in collaboration with less-agile, long term requirements planning processes. 2. Conway's law (MacCormack et al. 2012) also affects large agile development organizations. 3. Lehman's Law of Increasing Complexity (Lehman 1979) needs to be taken into account in large-scale agile development organizations. These implications are discussed in detail below.

1.) Our findings indicate that agile development organizations can effectively and efficiently work with medium and long term requirements planning processes and organizations that do not work using an agile life-cycle model. Subsequently, it is not necessary to completely disassemble and reorganize the whole organization in order to gain the benefits of agile development on the team level. However, based on our case, some adaptations are necessary in the team-level agile method and in the long-term requirements planning processes. On the team-level, the teams had to accept that they were assigned to develop a specific feature by the feature development process instead of negotiating the requirements selection with their product owner, as is the case in Scrum (Schwaber and Beedle 2002). The long-term planning process was adapted to not create precise long-term plans and expect that those plans are executed precisely. Instead, the release project planning process created constraints for the project, that is, the tentative release budget and feature content. The tentative constraints and feature ideas were then given as an input to the feature development process. Instead of predetermining the release contents at the beginning of the release project, the contents were determined based on what features were ready in time for the release. Although this reduced the predictability of the release scope, which had been quite high in the previous, plan-driven release planning process, it also increased the flexibility of the release scope and reduced the feature development lead time, which were major benefits from the new way of working.

2.) It seems that regardless of the agile ideal of anyone can do anything (Schwaber 2007; Beck and Andres 2004), Conway's law (MacCormack et al. 2012) should be observed also in large agile development organizations. An initial cross-case analysis between our case and existing literature (Moe et al. 2014; Larman and Vodde 2010; Leffingwell 2011; Augustine 2008; Heikkilä et al. 2015b) suggests that, especially when the system is large and technically complex, it may be beneficial to divide an agile development organization into technical areas that mirror the structure of the system. The teams can then be allowed to specialize in a technical area but should be generalists enough to be able to develop any part of that technical area and preferably in an end-to-end fashion. The work in and between the technical areas can be planned and coordinated in common release planning events (Heikkilä et al. 2015b), by technical area responsible people (Moe et al. 2014) or by teams of technical specialists and product owners and by communities of practice (Paasivaara and Lassenius 2014), as in our case. However, due to the limited amount of empirical research on large-scale agile development in

general, the results on the applicability of different planning mechanisms in different contexts are not conclusive.

3.) In early agile development texts, system level work is not explicitly covered and it was expected to be handled by the development team as a part of their daily work (Schwaber and Beedle 2002; Beck and Andres 2004). However, it seems that agile methods, at least in the context of large systems, cannot avoid Lehman's Law of Increasing Complexity (Lehman 1979). In large agile organizations, system level planning, documentation and problem solving needs to be explicitly taken care of. If these issues are not explicitly managed, there is some evidence that agile methods might cause increasing technical debt (Korhonen 2013). We suggest, that when adopting agile in the large-scale, initially the management of system-level work and technical debt might be handled by specialists such as technical product managers or architects together with product owners (Korhonen 2013; Moe et al. 2014). When the development teams have reached a good level of understanding about the agile principles of autonomy and self-organization, the responsibility can be transferred to self-organizing communities of practice that are assisted by the specialist (Paasivaara and Lassenius 2014).

6.5 Limitations and Threats to Validity

In the discussion about the validity of this research, we rely on the definitions of validity and reliability proposed by Yin (Yin 2009). The construct validity of a case study concerns the accuracy of the constructs created in the analysis to reflect the reality (Yin 2009). In order to increase the construct validity, we used triangulation of data sources and investigators (Yin 2009; Patton 2002). We triangulated the data sources by interviewing multiple persons holding each role in the case organization, when possible. We triangulated the investigators by having the first three authors perform the interviews, and by having almost all interviews conducted by two interviewers co-operatively. The interviews were coded by the first three authors and analyzed by the first author, but the second and the third author reviewed the analysis. Furthermore, the fifth author of this article was one of our key informants and he also reviewed the findings. The interview data collection method may induce many kinds of biases to the collected data (Shadish et al. 2001). Such biases could be mitigated by triangulating the interview data with data from project artifacts such as meeting memos and requirements documents. Unfortunately, due to confidentiality reasons, we did not have the possibility to study project artifacts qualitatively. However, the large number of interviewees and roles they had somewhat limits this threat to the construct validity. Based on the aforementioned triangulation, we can be quite confident that our findings accurately reflect the perceptions of the members of the case organization. However, we did not collect quantitative data on the changes in the development lead time, flexibility, planning efficiency or technical debt before and after the agile transformation. Subsequently, we cannot provide information about how much the transformation affected such indicators.

There was an approximately two year long gap between the first two interview rounds in 2011 and the third round in 2013. The data does not indicate significant differences between the team level requirements management practices in 2011 and 2013. Subsequently, combining the data on the team level requirements management practices from all interview rounds is not a significant threat to the construct validity of our findings on the team level practices.

Internal validity is concerned with the validity of causal relationships and it is only relevant to explanatory or causal results (Yin 2009). In the findings, we describe several causal relationships that were purported by the interviewees or otherwise identified by us. Due to the nature of a descriptive case study, the internal validity of the findings cannot be assessed accurately. There is a large number of threats to internal validity that cannot be removed in a descriptive case study, such as maturation and temporal ambiguity (Shadish et al. 2001). Thus, the findings of this study regarding causal relationships should be considered only tentative.

In order to increase the validity of our findings and to give feedback to the case organization, we presented the findings from the first and second interview rounds to the case organization in a feedback session conducted in December 2011 in Finland. All interviewees, as well as all interested case organization members were invited to the session. Approximately half of the interviewees participated in the session either locally or via a telecommunications system. Although there was a lot of discussion and questions, no corrections to our findings were necessary based on the feedback session.

The external validity of a case study concerns the context which the findings can be generalized to (Yin 2009), which is known as the analytical generalization of a case study (Yin 2009). The main question we need to address is as follows: If another organization adopted the processes and organizations described in this case, which characteristics must the organization share with our case for our findings to hold? Based on our study, we can create a hypothesis of the significant characteristics of the context. First, the system under development was large, multifaceted and technically demanding. Second, the number of development teams was relatively large. Third, there were three different levels of planning and management that had a different purpose, process and stakeholders. Different findings specific to each of these layers may be generalizable in isolation from the other layers if the result in question does not have significant dependencies to the other layers. On the other hand, the generalizability of the findings regarding the interfaces between the layers may demand that both layers are similar to those in our case. To increase the analytical generalizability of a descriptive single case study, the case study description must be detailed enough to allow analytical comparisons and synthesis with other studies on similar subjects (Yin 2009). We have striven to provide sufficient amount of details on the most significant findings to allow later comparisons and synthesis.

The reliability of research concerns the repeatability of the research (Yin 2009). If other researchers had conducted the same study, had they arrived at the same findings? The main threat to the reliability (Yin 2009) of this research is the variability in the data collection. The data collection was conducted using the general interview guide approach (Patton 2002), which introduced variability to the topics discussed in the interviews. However, the large number of interviewees and multiple interviewers allowed data source and investigator triangulation (Patton 2002) which increased the reliability of the findings.

7 Conclusions and Future Work

The applicability of agile methods to large development organizations has been a contentious issue in the practitioner literature. Our case study describes how agile methods were employed on a team level in a large organization, how the longer-term processes, the release project management process and the feature development process, interfaced

with the agile development organization and how requirements flowed between the different levels. Although the case organization had problems with adopting agile practices at a large scale and with the transition from a plan-driven to an agile way of working, overall the new way of working was considered a success. On the development team level, the main benefit was the increased motivation. On the release project management level, the main benefit was the significantly shorter development lead time and the increased flexibility of the development. On the feature development process level, the main benefit was the improved efficiency due to the incremental feature elaboration process. Especially, since large and agile telecommunications system software development organizations have not been widely studied beforehand, our in-depth case study allows comparisons to future research and different contexts to build more general theories regarding requirements management in large-scale agile organizations.

Our study was a qualitative, descriptive case study. Our approach was appropriate since the phenomenon we studied was not well known. The results indicated many changes to the performance of the case organization that were created by the agile transformation. However, our results are based on the perceptions of the case organization members. Now that many suggested benefits from the agile way of working have been identified, there is a need for further quantitative studies that study the actual effect size of these benefits to the overall performance of the organization. Our findings and the related work can be used as a basis for such quantitative studies. The proposed benefits of the agile way of working that need to be quantitatively studied include the following: effectiveness and efficiency of the selection of features, internal quality and technical debt, communication effectiveness and efficiency, the effects of the fast time-to-market, and the overall efficiency and effectiveness of the agile way of working.

Until recently, most literature on large-scale agile development was written by practitioners and consultants and the empirical evaluation of their proposed models and practices has been weak (Dikert et al. 2016). In addition to our case study, a few empirical studies have been recently published regarding the organizational structure of a large-scale agile development organization and the inter-team coordination and communication problems and practices. Different communication and coordination mechanisms, such as scrum-of-scrums, communities of practice and product owner teams, have been proposed in the research literature. This subject area would be a good target for a systematic literature review and meta-analysis to identify the contextual factors that affect the applicability of the different coordination mechanisms. Another interesting subject area is the study of the adaptations that had to be made to agile practices in order to scale agile methods to a large-scale organization.

Our case study is an additional contribution to the growing scientific knowledge base on agile development at scale. Our study extends this knowledge base by providing a detailed description of the requirements flow in the context of telecommunications software development in a large organization that employs an agile development method. In the future, we plan to concentrate on cross-case analysis of evidence from large-scale agile development in order to build generalizable knowledge and solid theoretical base for large-scale agile software development practice.

Acknowledgments We would like to thank Oy LM Ericsson Ab for making this study possible, all the anonymous interviewees for providing valuable contributions to this research and Kaisa Kettunen for reviewing the manuscript. This research was financially supported by TEKES (the Finnish Funding Agency for Innovation) as a part of the Cloud Software Finland program of DIGILE and the Need for Speed program of DIMECC.

Appendix: Interview Questions

1. Background
   - What is your role, background and experience at Ericsson?
   - What kind of training have you had on Lean and Agile software development? Would you have needed more training?
2. Agile and Lean Transformation
   - Why was the transformation started?
   - Explain the timeline of the transformation.
   - Why did you choose Scrum? Why Lean?
   - What comes from Scrum in your way-of-working? What comes from Lean in your way-of-working?
   - What is still left from the waterfall model?
   - What kind of training have you arranged? By whom? To whom?
   - What are you still planning to change / improve?
   - What is your general feeling of the transformation? Good / bad / challenges? How have you solved the challenges?
   - What would you have done differently? How?
   - Has Lean and Agile solved the challenges you wanted to solve by this change?
   - How have you measured the change? What have you exactly measured? How does this change look according to your measurements?
3. Organization structure
   - How does your organization look like? (draw a picture!)
   - How has the agile transformation affected the roles and responsibilities?
   - What roles do you currently have in your organization? Are the following roles and their responsibilities clear: Project Manager vs. Scrum Master vs. Product Owner vs. Proxy Product Owner vs. Program Manager? Are the roles roles clear in your organization? Are there challenges / improvement needs regarding these roles or their application?
   - Comment on the organization structure: Good? / Improvement needs? Do you still plan to change something regarding the organization structure?
   - How many persons do you have at the moment at each site? How many teams at each site? What are the team sizes at the different sites?
   - How are responsibilities divided between the different sites? Do you have different kind of tasks / roles at different sites? When did Hungary join in? Why?
   - How do you take care of the collaboration, communication and division of tasks between the sites? How does it work? Good? / Improvement needs?
   - Do different teams have different roles?
   - Feature teams: What kind of knowledge / "roles" do you have in each team? Good? / Challenges? What have you done to solve the challenges?

4. Coaching
   - External coaching: What kind of coaching / consulting have you had to support the transformation?
   - What kind of training have you arranged?
   - Have you had internal coaches? If yes, tell more about their role and tasks.
5. Team-level Scrum practices
   - Your own team: team size, what does your team do, connections to other teams?
   - Scrum practices of your team (explain all Scrum practices your team uses)
   - What is the length of your iterations? What is your opinion on the iteration length? Explain.
   - Cross-functional teams: How has the idea of the cross-functional teams worked out? Why? / Why not?
   - Tell about the communication inside your team. Good / bad / improvement needs?
   - How and when do you communicate with other teams?
   - Do you know enough on what is happening in the other teams / elsewhere in the project? Is there something that you would need to know more? What? Why?
   - How does the global distribution affect your daily work?
   - What is working well in you team / regarding your team practices? What are the biggest problems? What should be improved? How?
6. Improvement suggestions
   - Is there something that should be improved in this project?
   - How?
7. Tools
   - What are the most important tools that you use?
   - Tell a bit about each tool (for what is it used, who are using it, etc.) Good? / Bad?
8. Scaling agile / Cross-team coordination practices
   - What are your current scaling / cross-team coordination practices?
   - Scrum-of-Scrums? CoPs? Feature Owners? Tell about each practice.
   - How have you taken into account the global distribution in cross-team coordination? How do you handle communication? Tools used to support the communication and coordination? What kind of challenges have you experienced?
9. Measuring
   - What do you measure?
   - How do you measure quality? Productivity? How has this changed compared to the way of working before the transformation?
   - What should be measured? Why?
10. Product management
    - How is product management taken care of?
    - Where do you receive the requirements? Describe the whole flow from the customer request to the requirement being part of the product.
    - How is customer communication taken care of?
    - What does the Product Owner do? What does the Proxy Product Owner do?
    - Communication between the Product Owners/ Proxy Product Owners? Communication with the teams?
    - How is work divided between teams? Who does that?

- What are the current challenges of Product Management?

11. Opinions
    - How do you think that the Lean and Agile transformation has succeeded?
    - How do you think that Lean and Agile software development fits in your organization? What are the benefits is brings? What are the challenges? Has your opinion towards the Lean and Agile changed somehow during the transformation?
    - What advice would you give others considering the application of Lean and Agile to a similar situation?
    - What are your expectations towards our research?

References

Augustine S (2008) Managing Agile Projects. Prentice Hall Professional Technical Reference Upper Saddle River. NJ, USA

Bass JM (2015) How product owner teams scale agile methods to large distributed enterprises. Empir Softw Eng 20(6):1525–1557

Batista DM, Blair G, Kon F, Boutaba R, Hutchison D, Jain R, Ramjee R, Rothenberg CE (2015) Perspectives on software-defined networks: interviews with five leading scientists from the networking community. Journal of Internet Services and Applications 6(1):1–10

Beck K, Andres C (2004) Extreme programming explained : embrace change, 2nd edn. Addison-Wesley, Boston, MA, USA

Beecham S, Baddoo N, Hall T, Robinson H, Sharp H (2008) Motivation in software engineering: A systematic literature review. Inf Softw Technol 50(9-10):860–878

Brooks Jr. FP (1975) The Mythical Man-Month, Essays on Software Engineering

Cao L, Mohan K, Xu P, Ramesh B (2004) How extreme does eXtreme Programming have to be? Adapting XP practices to large-scale projects. In: Proceedings of the Hawaii International Conference on System Sciences, vol 37, pp 1335–1344

Chow T, Cao DB (2008) A survey study of critical success factors in agile software projects. J Syst Softw 81(6):961–971

Daft RL, Lengel RH (1984) Information richness: a new approach to managerial behavior and organizational design. Res Organ Behav 6:191–233

Damian D, Chisan J (2006) An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management. IEEE Trans Softw Eng 32(7):433–453

Dennis AR, Valacich JS (1999) Rethinking media richness: towards a theory of media synchronicity. In: Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences (HICSS-32)

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

Eckstein J (2014) Architecture in large scale agile development. In: Dingsøyr T, Moe N, Tonelli R, Counsell S, Gencel C, Petersen K (eds) Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation, Lecture Notes in Business Information Processing, vol 199. Springer International Publishing, pp 21-29

Elo S, Kyngäs H (2008) The qualitative content analysis process. J Adv Nurs 62(1):107–115

Fogelström ND, Gorschek T, Svahnberg M, Olsson P (2010) The impact of agile principles on market-driven software product development. J Softw Maint Evol Res Pract 22(1):53–80

Heikkilä VT (2015) Case studies on release planning in agile development organizations. PhD thesis, Aalto University, http://urn.fi/URN:ISBN:978-952-60-6046-0

Heikkilä VT, Paasivaara M, Lassenius C (2013a) ScrumBut, but does it matter? A mixed-method study of the planning process of a multi-team Scrum organization. In: O'Conner L (ed) Proceedings of the 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2013). IEEE Computer Society, pp 85–94

Heikkilä VT, Paasivaara M, Lassenius C, Engblom C (2013b) Agile Processes in Software Engineering and Extreme Programming, Lecture Notes in Business Information Processing. In: Baumeister H, Weber B (eds), vol 149. Springer, Berlin Heidelberg, pp 195–209

Heikkilä VT, Damian D, Lassenius C, Paasivaara M (2015a) A mapping study on requirements engineering in agile software development. In: Proceedings of the 41st Euromicro Conference on Software Engineering and Advanced Applications (EUROMICRO SEAA 2015), pp 199–207

Heikkilä VT, Paasivaara M, Rautiainen K, Lassenius C, Toivola T, Järvinen J (2015b) Operational release planning in large-scale Scrum with multiple stakeholders – a longitudinal case study at F-Secure Corporation. Inf Softw Technol 57:116–140

Hsieh HF, Shannon SE (2005) Three approaches to qualitative content analysis. Qual Health Res 15(9):1277–1288

Inayat I, Salim SS, Marczak S, Daneva M, Shamshirband S (2015) A systematic literature review on agile requirements engineering practices and challenges. Computers in Human Behavior 51. Part B:915–929

Jantunen S, Lehtola L, Gause DC, Dumdum UR, Barnes RJ (2011) The challenge of release planning. In: Proceedings of the Fifth International Workshop on Software Product Management, pp 36–45

Korhonen K (2013) Evaluating the impact of an agile transformation: a longitudinal case study in a distributed context. Softw Qual J 21(4):599–624

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

Larman C, Vodde B (2009) Scaling lean andamp; agile development: thinking and organizational tools for large-scale scrum. Addison-Wesley Upper Saddle River, NJ, USA

Larman C, Vodde B (2010) Practices for scaling lean & agile development: large, multisite, and offshore product development with large-scale Scrum. Addison-Wesley Upper Saddle River, NJ, USA

Lee EA (2002) Embedded software. In: Zelkowitz MV (ed) Advances in Computers, vol 56. Elsevier, pp 55–95

Leffingwell D (2011) Agile software requirements : lean requirements practices for teams, programs, and the enterprise. Addison-Wesley Upper Saddle River, NJ, USA

Lehman M (1979) On understanding laws, evolution, and conservation in the large-program life cycle. J Syst Softw 1:213–221

Lehtola L, Kauppinen M, Kujala S (2005) Linking the business view to requirements engineering: long-term product planning by roadmapping. In: Proceedings of the 13th IEEE International Conference on Requirements Engineering, pp 439–443

MacCormack A, Baldwin C, Rusnak J (2012) Exploring the duality between product and organizational architectures: A test of the "mirroring" hypothesis. Res Policy 41(8):1309–1324

Maglyas A, Nikula U, Smolander K (2012) What do practitioners mean when they talk about product management? In: Proceedings of the 20th IEEE International Requirements Engineering Conference (RE, vol 2012, pp 261-266

Maiden NAM (2012) Exactly how are requirements written? IEEE Softw 29(1):26–27

Moe NB, Dingsøyr T, Dybå T (2010) A teamwork model for understanding an agile team: A case study of a Scrum project. Inf Softw Technol 52(5):480–491

Moe NB, Šmite D, Šāblis A, Börjesson AL, Andréasson P (2014) Networking in a large-scale distributed agile project. In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '14), ACM, New York, NY, USA, pp 12:1–12:8

Noordeloos R, Manteli C, van Vliet H (2012) From RUP to Scrum in global software development: A case study. In: Proceedings of the Seventh International IEEE Conference on Global Software Engineering (ICGSE'12), pp 31–40

Paasivaara M, Lassenius C (2014) Communities of practice in a large distributed agile software development organization – Case Ericsson. Inf Softw Technol 56(12):1556–1577

Paasivaara M, Heikkilä VT, Lassenius C (2012a) Experiences in scaling the product owner role in large-scale globally distributed Scrum. In: Proceedings of the Seventh IEEE International Conference on Global Software Engineering (ICGSE 2012), pp 174–178

Paasivaara M, Lassenius C, Heikkilä VT (2012b) Inter-team coordination in large-scale globally distributed Scrum: do Scrum-of-Scrums really work? In: Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM'12), ACM, New York, NY, USA, pp 235-238

Paasivaara M, Lassenius C, Heikkilä VT, Dikert K, Engblom C (2013) Integrating global sites into the lean and agile transformation at ericsson. In: Proceedings of the IEEE 8th International Conference on Global Software Engineering, pp 134–143

Paasivaara M, Väättänen O, Hallikainen M, Lassenius C (2014) Supporting a large-scale lean and agile transformation by defining common values. In: Dingsøyr T, Moe NB, Tonelli R, Counsell S, Gencel C, Petersen K (eds) Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation: XP 2014 International Workshops, Revised Selected Papers, Springer International Publishing, Lecture Notes in Business Information Processing, vol 199, pp 73–82

Patton MQ (2002) Qualitative research and evaluation methods, 3rd edn. Sage Publications, Thousand Oaks, CA, USA

Pink DH (2011) Drive: The Surprising Truth About What Motivates Us. Riverhead Books

Rautiainen K, Lassenius C, Sulonen R (2002) 4CC: A framework for managing software product development. Eng Manag J 14(2):27–32

Schwaber K (2007) The enterprise and Scrum. Microsoft Press Redmond, WA, USA

Schwaber K, Beedle M (2002) Agile software development with Scrum

Shadish WR, Cook TD, Campbell DT (2001) Experimental and quasi-experimental designs for generalized causal inference. Houghton Mifflin, Boston, MA, USA

Svahnberg M, Gorschek T, Feldt R, Torkar R, Saleem SB, Shafique MU (2010) A systematic review on strategic release planning models. Inf Softw Technol 52(3):237–248

Taramaa J, Seppänen V, Mäkäräinen M (1996) From software configuration to application management—improving the maturity of the maintenance of embedded software. J Softw Maint Res Pract 8(1):49–75

Verner JM, Babar MA, Cerpa N, Hall T, Beecham S (2014) Factors that motivate software engineering teams: A four country empirical study. J Syst Softw 92:115–127

Vähäniitty J (2012) Towards agile product and portfolio management. PhD thesis, Aalto University, http://urn.fi/URN:ISBN:978-952-60-4506-1

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 (eds) Product-Focused Software Process Improvement, Lecture Notes in Computer Science, vol 7983. Springer Berlin Heidelberg, pp 35-49

Yin RK (2009) Case study research : design and methods, 4th edn. Sage Publications, Thousand Oaks, CA, USA

Ville T. Heikkilä is a postdoctoral researcher at Aalto University, Finland. His specific research interests include agile and lean methods, requirements engineering, DevOps and continuous software development. He also conducts research on serious games for software engineering education. He has a D.Sc. degree from Aalto University, Finland.

Maria Paasivaara is a research fellow at Aalto University. Her research interests include agile and lean software engineering, scaling agile to large and globally distributed projects, global software engineering, and software engineering education. She has a D.Sc. degree from Aalto University.

Casper Lassenius is an associate professor of software engineering at the Aalto University, Finland. His research is conducted empirically in close collaboration with industrial partners, trying to minimize the gap between industry and academia. His more specific interests include software product development, agile methodologies, global software engineering, and software testing and quality assurance. He has a PhD in Computer Science from Helsinki University of Technology, Finland.

Daniela Damian is a Professor of Software Engineering in University of Victoria's Department of Computer Science, where she leads research in the Software Engineering Global interAction Laboratory (SEGAL, thesegalgroup.org). Her research interests include Empirical Software Engineering, Requirements Engineering, ComputerSupported Cooperative Work. Her recent work has studied the developers' sociotechnical coordination in large, geographically distributed software projects, as well as stakeholder management in large software ecosystems. Daniela has served on the program committee boards of several software engineering conferences, and recently was the Co-Chair for the Software Engineering in Society Track at ICSE 2015. She is currently serving on the editorial boards of Transactions on Software Engineering, the Journal of Requirements Engineering, is the Requirements Engineering Area Editor for the Journal of Empirical Software Engineering, and the Human Aspects Area Editor for the Journal of Software and Systems.

Christian Engblom is a member of the Ericsson Finland R&D management team. After joining Ericsson in 1979 he has held roles of project manager, product manager and line manager within the R&D, as well as the market unit areas. Currently he holds a position as driver for the Lean and Agile transformation within the R&D organization in Finland and is actively involved in Lean & Agile change wave going through the entire Ericsson R&D world. His professional interests include agile and lean software development.