FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Information Systems

Empirical Studies to Identify Challenges and Probe Good Practices in the Adoption of Scaled Agile Methods in the Field of Vehicle Dynamics Development of an OEM

Clara Lea Buchholz

FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Information Systems

Empirical Studies to Identify Challenges and Probe Good Practices in the Adoption of Scaled Agile Methods in the Field of Vehicle Dynamics Development of an OEM

Empirische Studien zur Identifizierung von Herausforderungen und Erprobung bewährter Praktiken bei der Einführung skalierter agiler Arbeitsweisen im Bereich der Fahrdynamikentwicklung eines OEMs

Author:	Clara Lea Buchholz
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	M. Sc. Ömer Uludağ
Date:	August 14, 2019

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, August 14, 2019 Clara Lea Buchholz

Abstract

Since 17 scientists published the Agile Manifesto in 2001, agile projects have been continuously evolving and their success attracted attention. The approach to install agile development proposed lean processes, the continuous improving of the product, and more flexibility. Even though in the first place, the term agile is just a set of 12 principles and four values. With these in mind, various methodologies, practices, and frameworks were introduced and adapted in the last decades. With their usage, new concerns arose. Agile development was not only installed in individual and co-located teams for which it was initially designed, but was also adapted in large organizations. Therefore, the transformation from traditional to agile development was experienced as more challenging in large than in small organizations. Additional concerns came up in large-scale agile development and were addressed by novel practices, which required examination. Research in this domain is still scarce, while gaining relevance, especially within transformations. In particular, the demand for case studies increases. Therefore, we investigated the individual adoption of agile development in a large department producing software in the field of vehicle dynamics development. Additionally, we analyzed the concerns in the different adoption phases. Overall, we reinvestigated 55 concerns from previous literature and found 27 new concerns. The recurring concerns were addressed by several good practices. To document these probed good practices in a structured manner, the large-scale agile development pattern language developed by our chair was used. 17 pattern candidates are demonstrated in this thesis, divided into Principles, Coordination Patterns, Methodology Patterns, Viewpoint Patterns, and Anti-Patterns. To fulfill our empirical research approach, we conducted 14 interviews with scrum masters, product owners, developers, and managers and observed their work for five months. We were able to analyze not only the temporal occurrence, but also the role-specific experiences with the documented concerns and good practices as well as relationships between them.

Contents

Outline of the Thesis

CHAPTER 1: INTRODUCTION

The first chapter offers the motivation of this thesis to the reader as well as explains research objectives, which include three research questions and how those will be answered.

CHAPTER 2: FOUNDATIONS

Divided into four parts, this chapter explains the foundation of this thesis. It comprises first announcing patterns, second introducing agile development at all, followed by large-scaled agile development and last stating the transformation from traditional development to large-scale agile development.

CHAPTER 3: RELATED WORK

The analysis of related literature regarding concerns, success factors and patterns in large-scale agile development and during transformations from traditional to large-scale agile development is presented in this chapter.

CHAPTER 4: CASE STUDY

This chapter explains design and description of the case study. Furthermore, the procedure of the conducted transformation is demonstrated and results about applied scaling practices are presented.

CHAPTER 5: OBSERVING AND IDENTIFYING RECURRING CONCERNS AND GOOD PRACTICES

In this chapter all relevant results are summarized. First, the recurring concerns are listed and explained. Furthermore, used good practices are clarified and registered as pattern candidates.

CHAPTER 6: DISCUSSION

This chapter discusses the key findings and limitations of this thesis.

CHAPTER 7: CONCLUSION

The last chapter consists of a summary of this thesis and a possible outlook for future work.

1. Introduction

This first chapter includes the subject matter which motivated us for this thesis. Next, research objectives with our three research questions are described: First how the large-scale agile transformation takes place at the case organization, second what concerns within large-scale agile transformation at the Original Equipment Manufacturer (OEM) arose, and third what good practices were used to address the observed concerns within large-scale agile transformation at the OEM. Moreover, our research approach explains how we conducted our empirical study.

1.1. Motivation

Agile development has evolved, since the publication of the Agile Manifesto [9] in 2001 with its 12 principles of agile software and four agile values. The manifesto defined a new way of developing software with focus on individuals, customers as well as developers, their interaction, their products, and flexibility. Nevertheless, existing procedures are still appreciated [9]. From these values and principles several practices and methodologies arose [22]. Although, original agile methods were first designed for small, individual, and co-located teams [22], large departments have also shown interest in trying agility in a large-scale development [49]. Traditional software development methods have sometimes failed in quickly changing business environments, because of their inflexibility [49]. Agility proposed more flexibility [34] through a new mindset [18]. While adopting agility in larger departments new challenges arose [18, 35, 49, 67]. Nevertheless, organizations requested to explore new potential in software development. A higher demand for coordination [57] as well as a challenging collaboration with non-agile departments [27] were expectable new concerns. As the domain gained relevance, more research was done [23], but general valid academic work is still limited, especially regarding large-scale agile development [1]. Special research about the concerns coming up during large-scale agile transformations was done but is rare [21, 67]. Additional experience reports about ways to address challenges encountered in large-scale agile development are also scarce [17,66]. After theoretical research was done, the demand for practical research is increasing. This is the reason why this thesis aims to analyze the transformation phase from a traditional matrix organization (skill-oriented) to a cross-functional large-scale agile organizational structure. Additionally, other concerns and modes in which they were addressed during the transformation will be analyzed. The collection of data was done by observing the development department of vehicle dynamics and by conducting 14 interviews with different stakeholders from the department. Also, provided data was taken into account. During empirical research of this thesis, several good practices were investigated at the case organization. The intention was to help other departments and external organizations in the adoption of scaled agile methods with experiences from the case study organization as well as adding value to academic research with a new case study.

1.2. Research Objectives

For this work empirical studies have been conducted, to identify concerns and probe good practices during the adoption of scaled agile methods in a specific case organization. To cluster the investigated experiences, three research questions have been defined:

> **Research Question 1:** How does the large-scale agile transformation take place at the case organization?

The transformation of a software development department of an OEM in the field of vehicle dynamics development from utilizing traditional software development methods to employing scaled agile development methods took place in the observed time. The first research question intends to examine how this transformation was implemented at the case organization, what framework was used, and how the large-scale agile approach was conducted. The investigation was prepared by studying existing literature of other case studies during the transformation of organizations and conducted by both, observing the case organization and performing interviews.

> **Research Question 2:** What are concerns within the large-scale agile transformation at the OEM?

The second research question aims to find out what concerns arose during the transition to large-scale agile development. With respect to concerns which already have been analyzed in literature, different stakeholders at the case organization were requested to share their experiences with concerns in their daily business. Additionally, agile teams were observed during their meetings to identify more concerns.

> **Research Question 3:** What are good practices to address the observed concerns within the large-scale agile transformation at the OEM?

Since the first research question explained how the transformation took place and the second research question led to issues during the transformation, the last research question seeks to find out how to solve these issues. Based on the results of the first and second research question we fulfilled our further data collection (see Section 4.1). We identified different practices as good practices and documented them as pattern candidates. They were used in agile teams to shape agile development. The pattern candidates were clustered in a pattern catalog to be used for scientific reasons and by other organizations undergoing the same transformation. Patterns as well as pattern candidates will be explained in detail in Section 2.1.

1.3. Research Approach

This thesis aims to answer the three research questions explained in Section 1.2. According to Robson [53] four different purposes for research exist: examination, description, explanation, and improvement. Our research purposes to fulfill all four: first we explore the current state of the art at the case organization, second we describe the situation, third we try to find explanations and links for the arising concerns, and fourth we improve the situation by probing good practices. This thesis was structured according to the paradigm of design science by Hevner et al. [33] (see Figure 1.1). Mentioned scientists summarized seven suggestions on how to conduct a design-science research guideline (see Table 1.1).

Guideline	**Description**
Guideline 1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Table 1.1.: Design-Science Research Guidelines by Hevner et al. [33]

We satisfy the guidelines in the following way:

1. *"Design as an Artifact"* [33]: The artifacts developed by this thesis are three main scaling practices (see Chapter 4.2), new findings on concerns (see Chapter 5.1), as well as the catalog of pattern candidates found at the case organization (see Chapter 5.2 and Appendix B). Our results will be used for the Large-Scale Agile Pattern Language of the chair for software engineering for business information systems (cf. [65]).

2. *"Problem Relevance"* [33]: The objectives have been explicated in Section 1.2: Focused on the process of transformation concerns, success factors, and good practices have been discovered and investigated.

3. *"Design Evaluation"* [33]: The design evaluation can be found in Section 4.1: The case study design was oriented towards Lethbridge et al. [40]. Especially data collection techniques by Lethbridge et al. [40] were applied. Said scientists clustered data collection for software engineering in three degrees of human intervention: first direct involvement of stakeholders, second indirect involvement of stakeholders, and third study of work artifacts only. The first and third degree were used for this thesis.

4. *"Research Contributions"* [33]: The design artifact (see first item), the foundation (see Chapter 2), and the methodology (explained in this chapter and Chapter 4.1) summarize the contribution of this work, which is merged in Section 6.1 to present the key findings.

5. *"Research Rigor"* [33]: We aimed the rigor conduction of the artifacts by demonstrating the realization of the "behavioral theories" [33] (see related work in Chapter 3) and "empirical work" [33] (see our case in Chapter 4 as well as our findings on concerns and pattern candidates in 5).

6. *"Design as a Search Process"* [33]: The iterative process of observing the case organization and using relevant literature formed the search process of our data.

7. *"Communication of Research"* [33]: As this thesis was conducted with a partner company, the last guideline is as follows satisfied: The deliverable conducted through this work was presented to the scientific (technology-oriented) chair of Software Engineering for Business Information Systems in the same way as to management-oriented and technology-oriented stakeholders at the case organization.

As Figure 1.1 illustrates, we use the case organization as environment (see Chapter 4) and on the other hand, our literature research (see Chapter 2 about the foundations of this thesis and Chapter 3 about related work) for knowledge base. As already Flyvbjerg [26] discussed, to demonstrate theoretical knowledge a case study can be helpful and spend generalizability if linked to theoretical knowledge. This is why this thesis conducts a study

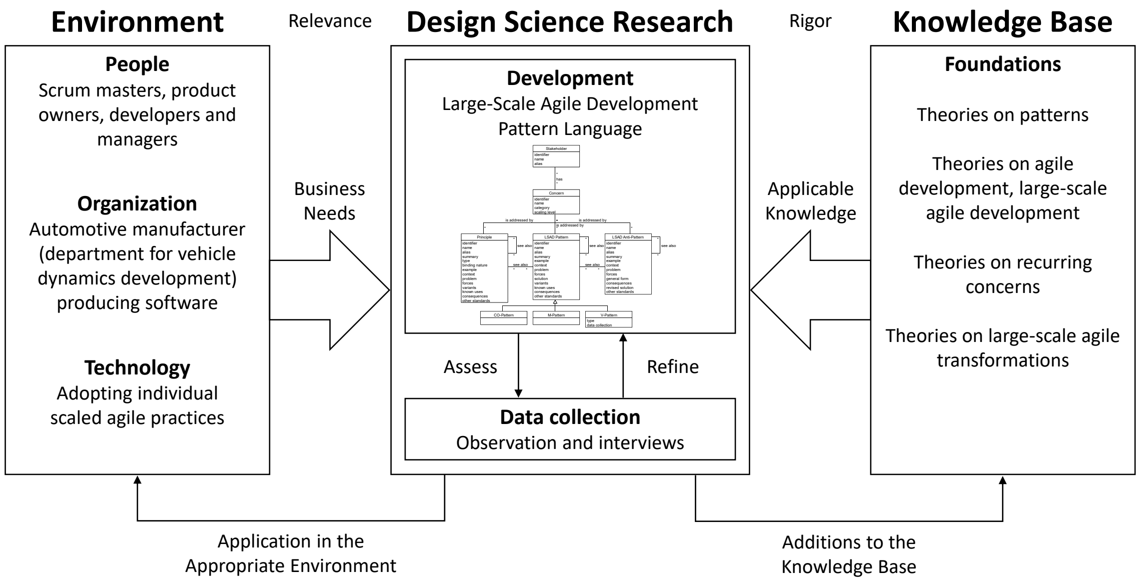
Environment
Relevance
Design Science Research
Rigor
Knowledge Base
People
Scrum masters, product owners, developers and managers
Organization
Automotive manufacturer (department for vehicle dynamics development) producing software
Technology
Adopting individual scaled agile practices
Business Needs
Development
Large-Scale Agile Development Pattern Language
Assess
Refine
Data collection
Observation and interviews
Applicable Knowledge
Foundations
Theories on patterns
Theories on agile development, large-scale agile development
Theories on recurring concerns
Theories on large-scale agile transformations
Application in the Appropriate Environment
Additions to the Knowledge Base

Figure 1.1.: Design-Science approach of this thesis according to Hevner et al. [33]

to identify concerns, probe good practices from literature in practice, and find new ones. A qualitative study with semi-structured interviews and unstructured interviews according to Runeson and Höst [55] has been performed with different stakeholders at the case organization to explore the current state of the organization and get detailed insights in the personal experience of different associates. Detailed information about these interviews at the case organization and the applied data collection techniques by Lethbridge et al. [40] can be found in Section 4.1.

The next chapter lists required terms and concepts regarding the subject matter, whereas Chapter 3 introduces related scientific work. Chapter 4 offers an overview on design and description of the case study as well as the results regarding the first research question (**Research Question 1:** *How does the large-scale agile transformation take place at the case organization?*). Chapter 5 includes all relevant results regarding the second and third research question (**Research Question 2:** *What are concerns within the large-scale agile transformation at the OEM?*; **Research Question 3:** *What are good practices to address the observed concerns within the large-scale agile transformation at the OEM?*). Findings about concerns as well as good practices are presented and explained. The sixth chapter discusses the key findings and limits of this thesis while the seventh summarizes our research.

2. Foundations

This chapter clarifies the relevant terms and concepts our thesis deals with. First, patterns are introduced, which are the deliverable of this case study. As agile development is the base of this thesis, in the following sections a definition of agile development methods and the agile framework Scrum will be given. Afterwards, scaled agile development methods will be defined, their need will be analyzed, and special concerns will be demonstrated. Third and fourth, success factors and common frameworks to address the concerns and benefits from the success factors in scaled agile development will be documented. After understanding scaled agile development methods, the last part of this chapter explains how they can be adopted.

2.1. Patterns

While some decisions in software engineering are matchless, others provide solutions for concerns which are not unique, but recurring. These can be written down as patterns (cf. [17, 28]). The use of patterns can save time and energy for unique problems [8], especially in large-scale [10], and is appreciated by scientists, engineers, and managers [12]. Patterns can be used to describe different complex things according to the Alexandrian Form [3]. Said scientist outlined patterns as instructions for "a relation between a certain context, a problem, and a solution" [3]. Many others referenced to the Alexandrian Form in their explanations of patterns, for example Coplien [16], the Gang of Four [28], and Wellhausen and Fiesser [72]. Main element of patterns is a solution of concerns in a specified context [28]. As Riehle [52] explained, patterns can vary in their granularity and presentation.

Different ways to document patterns exist, and Harrison [31] structured them into several steps with various ways to go through. Important for him was the core idea including a detailed explanation, consequences, and forces that guide to the solution [31], all easy to understand for the target audience [31]. Patterns can be named differently, for example after a fitting metaphor or the solution they provide [42]. To identify a pattern and to avoid misunderstandings an identifier can be used [66]. Wellhauser and Fiesser [72] summarized five related main sections which can be expanded to more: "Context, Problem, Forces, Solution, [and] Consequences" [72]. To simplify the writing of patterns another running order can be used. Said scientists recommended to first imagine the solution, second the description of the problem, which is followed by the consequences, then the forces, and last the context [72]. As summary of a pattern provides a short overview and easier understanding [42].

Patterns can be categorized into different types. Relevant for this thesis are Principles, Coordination Patterns, Methodology Patterns, Viewpoint Patterns, and Anti-Patterns according to the large-scale agile pattern language (see the conceptual model in Figure 2.1) by Uludağ et al. [66]:

- **Principles** are "enduring and general guidelines that address given concerns by providing a common direction for action" [66].
- **Coordination Patterns** (CO-Pattern) state instruments used to address recurring concerns.
- **Methodology Patterns** (M-Pattern) represent clear processes that help to solve arising concerns.
- **Viewpoint Patterns** (V-Pattern) document visualizations that foster solving concerns.
- **Anti-Patterns** are pattern that should be avoided, because they picture possible faults. In their description alternative solutions are presented.

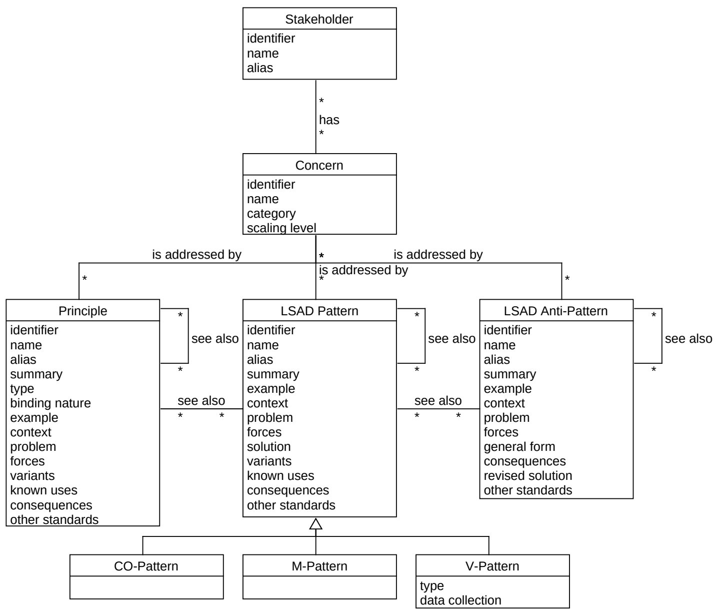

Figure 2.1.: "Conceptual model of the large-scale agile development pattern language" by Uludağ et al. [66]

In accordance with Uludağ et al. [66] every element in their pattern language (see Figure 2.1) has an *identifier* as well as a *name*. Stakeholders, principles, large-scale agile development patterns, and anti-patterns additional store *aliases*. For principles, large-scale agile development patterns, as well as anti-patterns a short *summary* concludes the *solution*. The *example* outlines a possible use, the *context* illustrates a situation in which the practice can be used, and the *problem* demonstrates the problem itself. *Forces* describe difficulties during the solving of the concern and additional *consequences* regarding benefits and liabilities are listed [66]. *Variants* and relations (*see other*) to other elements of the large-scale agile development pattern language are documented, if existing [66]. As the *known use* in this thesis is always the same, this section is redundant for us.

According to Coplien [16], every documented pattern must recur not less than three times as validation. As this thesis is based on the empirical study which was conducted at a specific organization, the rule of three cannot be met. Therefore, pattern candidates instead of patterns are documented in Chapter 5.

2.2. Agile Development

Previously, the explanation of patterns provided general information about their use in software engineering. The next section outlines principles and values of agile development. Furthermore, the framework Scrum will be explained, as application area for agile development.

2.2.1. Principles and Values

In software development different work methods are used: sequential, iterative or agile models. The first two are referred to as traditional software lifecycle models [6]. The sequential models are for example the Waterfall Model or the V-Model, while iterative models are the Spiral Model, the V-Model XT, and others (cf. [11]). The third category are lean agile models, for example Scrum, Kanban, and Extreme Programming (cf. [6,11]). As traditional software development methods tend to fail sometimes in a quickly changing business environment [49], whereas agility and flexibility are more important these days, agile development methods offer alternative solutions [34]. Furthermore, agile software development methods overall have gained more relevance in the last decades [22].

Agile software development comprises more than just work methods. It is a development philosophy [48]. The agile mindset started with the Agile Manifesto in 2001: 12 principles of agile software and four agile values were defined and published [74]. The Agile Manifesto aimed to "allow software teams to work quickly and respond to change" [74]. It includes the following 12 principles [9]:

- "Our highest priority is to satisfy the customer through early continuous delivery of valuable software."
- "Welcome changing requirements, even late in development. Agile processes harness change for the customers competitive advantage."
- "Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale."
- "Business people and developers must work together daily throughout the project."
- "Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done."
- "The most efficient and effective method of conveying information to and within a development team is face-to-face conversation."
- "Working software is the primary measure of progress."
- "Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely."
- "Continuous attention to technical excellence and good design enhances agility."
- "Simplicity - the art of maximizing the amount of work not done - is essential."
- "The best architectures, requirements, and designs emerge from self-organizing teams."
- "At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."

Additionally, mentioned scientists specified the following four values [9]:

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

Software developers should pursue to develop an increment of high benefit to their client [9]. Agile software development techniques can help them as simple tools in their processing [74]. The popular framework Scrum provides several techniques [61] and is explained in the next section.

2.2.2. Scrum

As written before, agile frameworks offer different techniques to work with. One of the most popular is the Scrum framework. It does not instruct how software is developed, but provides processes to develop and to deal with change [51]. The focus is on well-regulated teamwork instead of techniques. The team itself can select useful techniques [51]. In center of this teamwork are three main roles, three artifacts, and iterations (so-called sprints) with four meetings explained by Schwaber and Beedle in the "Agile Software Development with Scrum" [60] and summarized in the following subsections.

Roles

Scrum provides three roles [60, 61]:

1. The **Product Owner** defines the product and is responsible for the increment.

2. The **Scrum Master** is responsible for the process and resolves impediments.

3. The **Development Team** realizes the product increment. Wintersteiger [75] explained that there are three important characteristics for development teams:
    a) Self-organized: There is a clearly defined goal and the team works together to succeed.
    b) Skill-oriented: All needed competencies to achieve the goal are available in the team.
    c) Responsible: The team is collective responsible for the success or failure of the project.

Artifacts

Scrum is using three artifacts [60, 61]:

1. The **Product backlog** is a list of requirements for the whole product. It is a collection of items (re)prioritized by the product owner.

2. The **Sprint backlog** is a list of requirements and tasks for one iteration (sprint).

3. The **Potentially Shippable Product Increment** (PSPI) is a release which is reviewed with the product owner and contains the deliverables of the current sprint.

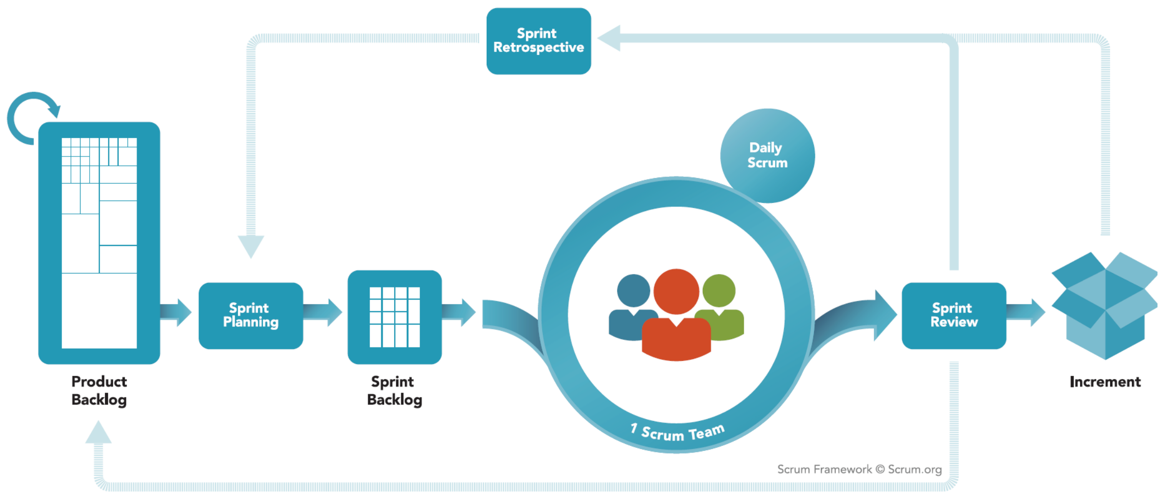

Figure 2.2.: Scrum Framework [61]

Meetings in a Sprint

First the project is planned in a **project kickoff meeting** to create and prioritize the product backlog. One project contains several sprints. **Sprints** are increments in which one PSPI is produced by a development team. The length of a sprint is variable and can be scheduled individually. Usually a sprint lasts between one and four weeks. Every sprint starts with a **sprint planning meeting** to create the sprint backlog. During the sprint, every day a stand-up meeting takes place called **daily scrum meeting**. The members of the development team share their status, impediments, and promises for the next day. Every sprint ends with a **sprint review** to demonstrate the realized backlog items to their product owner and stakeholders. The acceptance criteria (conditions that a software product must satisfy to be accepted by a stakeholder) for every single task are checked and after fulfilling all, the tasks and finally the sprint is closed. After every sprint the team members meet with their scrum master to inspect the previous sprint and plan improvements in their working style for the next sprint in a **sprint retrospective** (cf. [51, 60, 61], see also Figure 2.2).

2.3. Large-Scale Agile Development

The demand for agile development occurred not only in individual teams, but also in larger organizations. To outline their use of agility, the first part of this section defines large-scale agile development. The following explains arising concerns and success factors in large-scale agile development. Lastly, we provide an overview about frameworks especially for scaling agile development.

2.3.1. Definition

Large-scale agile development is the use of agile methodologies in several teams of one organization regarding one product [19]. Agile methods, like Scrum, were initially constructed for small, individual, and co-located teams, therefore, new concerns came up when used in large IT organizations [48]. Small-scale projects are performed by one team, large-scale by two to nine teams, and very large-scale projects by more than 10 teams [5,19]. The complexity of agile development increases with the size of the organization [23]. Therefore, large-scaling gained relevance fast and the need for specific solutions escalated [21]. Thus, scaled agile development methods were developed in different organizations and different forms [22].

2.3.2. Concerns in Large-Scaled Agile Development

We aim to discover concerns which arose at the case organization. Therefore, known concerns in large-scaled agile development will be explained.

Paasivaara [46] discovered that the cooperation between teams which still worked with traditional methods and agile teams can lead to a concern. Such situations appear more often in large-scale agile development [46]. Furthermore, others identified the interaction between agile and non agile teams as concern in scaled agile development (cf. [58]). Scheerer et al. [58] mentioned the coordination in large-scale agile development in general, not only between agile and non agile teams, as more important. Abrar et al. [2] identified the influence of agile work to management as further concern. The agile development influenced the processes, consequently a loss of calculable project time occurred. This was experienced as exhausting for developers [2].

More comprehensive, in 2016 Dikert et al. [18] reported 35 challenges clustered into nine categories found through a systematic literature review. Later, Uludağ et al. [67] identified and listed 79 challenges assigned to 11 categories in 2018, with the aim to refine the list by Dikert et al. [18] with more information. Therefore, stated scientists outlined insightful which stakeholder is related to which concern and where those were found. In 2019, Uludağ and Matthes [70] extended this list after conducting and analyzing interviews with industry experts through empirical research. More about known concerns itself can be found in Chapter 3, where a detailed view about named scientific activities is given. This thesis aims to demonstrate found concerns by their occurrence in the case organization, and if new concerns are observed, to add them to the list.

2.3.3. Success Factors in Large-Scaled Agile Development

To get a complete overview also the known success factors have to be explained. Dikert et al. [18] found 29 success factors especially during large-scale agile transformation and clustered them into 11 categories [18] (listed by most frequent identified):

1. "Choosing and customizing the agile approach
2. "Mindset and alignment"
3. "Management support"
4. "Training and coaching"
5. "Piloting"
6. "Team autonomy"
7. "Requirements management"
8. "Commitment to change"
9. "Leadership"
10. "Communication and transparency"
11. "Engaging people"

In all those categories we can find factors, which were identified as success factor in the observations by Dikert et al. [18]. For example for category 1: "Choosing and customizing the agile approach" Dikert et al. [18] called "Customizing the agile approach carefully [,] Conform to a single approach [,] Map to old way of working to ease adaption [, and] Keep it simple" [18] as success factors.

Kalenda et al. [35] also identified different success factors, first during a literature review, and clustered them in a different scheme [35]:

1. "Acquire knowledge"
2. "United view on values and practices"
3. "Tools and Infrastructure"
4. "Solid engineering practices"
5. "Careful transformation"
6. "Teamwork support"
7. "Executive sponsorship"

Afterwards, Kalenda et al. [35] performed an empirical study and limited their list to four main success factors based on their study. "United view on values and practices" [35] (see Item 2) and "executive sponsorship" [35] (see Item 7) have already been found before and were now demonstrated. Additionally, they found "company culture and prior agile and lean experience" [35] as two new success factors.

The studies by Dikert et al. [18] and Kalenda et al. [35] differed in some points. The four final success factors by Kalenda et al. [35] can be found in slightly different versions in the list by Dikert et al. [18] as shown in Table 2.1. The table shows relations between both, but not similarities. Especially "company culture" [35] is not easy to match. Dikert et al. [18] solved equal concerns with the category "Mindset and Alignment" [18]. Furthermore, they analyzed "Choosing and customizing the agile approach" [18] (see Item 1), as most recurring in their studies, while Kalenda et al. [35] did not mentioned this one at all.

Dikert et al. [18]	Kalenda et al. [35]
United view on values and practices	Mindset and Alignment
Executive Sponsorship	Management support
Company culture	(Mindset and Alignment)
Prior agile and lean experience	Piloting

Table 2.1.: Relations between the categories of success factors by Kalenda et al. [35] and Dikert et al. [18]

Larman and Vodde [38] guided their reader through the scaling of agile development in their book, pointing to a various range of success factors. Most of them have in common that they were not only regarding the content, but also the mindset. They referred to the fulfillment of the agile mindset, e.g. during their explanation about Communities of Practice [38], which were also called as a success factor by Paasivaara and Lassenius [47]. Additionally, the importance of an open and passionate community of participants for Communities of Practice was recognized [47]. Paasivaara and Lassenius [47] also highlighted to live the agile mindset. More precisely, to not just change the name of a community or group, but also the functionality, structure, and working of a community for being agile successful (cf. [38]).

As outlined in Section 2.1 common solutions can be illustrated as patterns which represent good practices to address concerns [16]. Uludağ et al. [66] identified some pattern for their found concerns and categorized them in the previously explained structure of Principles, Coordination Patterns, Methodology Patterns, Viewpoint Patterns, and Anti-Patterns.

2.3.4. Scaling Agile Frameworks

Different frameworks can help to scale agile development and solve upcoming concerns. Some are just loose suggestions, others contain strict regulations. A short summary about three frameworks extending the previously explained Scrum framework for the large-scale agile development follows: Large-Scale Scrum, Scrum@Scale, and Scaled Agile Framework.

In 2005 Large-Scale Scrum (LeSS) was adopted first, descriptions of the framework followed in 2009 [38]. The LeSS framework scales the use of Scrum to more teams. Development teams are called Feature Teams in LeSS, because of their characteristics [38]. All Feature Teams share "a single product backlog, the same definition of done [,] synchronized [s]prints to a potentially shippable product after each sprint, and a single product owner" [48], moreover, they have common meetings: sprint planning, review, and retrospective are conducted together with all Feature Teams. If needed, Feature Teams can add individual meetings without other Feature Teams [38]. LeSS was designed for up to eight individual teams. For more teams LeSS Huge is recommended [38], which is an additional extension.

Scrum@Scale coordinates the use of Scrum to more than one Scrum team. A set of teams working on the same product is called a Scrum of Scrums (SoS). Depending on the size of the organization, also a Scrum of Scrums of Scrums can be installed from a crowd of SoS. Scrum@Scale divides a Scrum Master Cycle (responsible for the "how") and a Product Owner Cycle (responsible for the "what") [62].
The Scrum Master Cycle: Additional to the roles and meetings in Scrum, the Scrum@Scale framework provides the role of a Scrum of Scrum Master and a Scaled Daily Scrum which is a daily scrum meeting with representatives from every team in the SoS (cf. [62]).
The Product Owner Cycle: The Product Owner Team includes all product owner who are responsible to prioritize and generate tasks for one common backlog. Every SoS has its own Product Owner Team. Additional to the roles in Scrum, the Scrum@Scale framework provides the role of a Chief Product Owner, who has the scaled responsibilities of a regular product owner [62].
The Scrum Master Cycle and the Product Owner Cycle interact with each other. They meet on the "Team-Level Process" [62] as well as regarding the product and its release [62].

The Scaled Agile Framework (SAFe) scales agile practices different. SAFe divides the development in different levels: portfolio, large solution, program, and team. First the team level includes activities, processes, events, and roles which take place for the team. While second the program level includes roles and activities regarding the Agile Release Train (ongoing team of agile software development member working together in a train on an ongoing program). The third one is the large solution level, where large and complex solutions are constructed with the required roles, artifacts, and processes. The last one is the portfolio level covering principles, practices, and roles to the set of development value streams. The foundation of the SAFe are principles and values, strengthening the lean-agile mindset [39].

LeSS, Scrum@Scale, and SAFe are just three frameworks for scaling agile development which were picked because of their close relation to the Scrum framework. Moreover, all three were designed for scaling on the same level, as needed for the case organization, and LeSS as well as SAFe were described as popular by Uludağ et al. [69]. Many others exist, e.g. the Disciplined Agile Delivery which combines techniques of the Extreme Programming, Unified Process, Kanban, and Agile Modeling to expand Scrum [4], and the Nexus framework by Schwaber [59] (signee of the Agile Manifesto in 2001). The latter one extends the Scrum framework to scale it with an additional Nexus sprint backlog, advanced meetings, an integrated increment, and more, but is limited to nine teams [59].

2.4. Large-Scale Agile Transformation

The transformation of an organization from traditional to agile software development is a concern on its own. Fuchs and Hess [27] recommended that "a large-scale agile transformation can be interpreted as an episodic, socio-technical change" [27]. The adoption of agile development is more than the adoption of a set of agile practices. Most important is the mindset change of all involved employees and their stakeholders, independent of the applied framework [24]. To avoid that only "old wine in new bottles" [41] is served, which Hilkka et al. [41] implied, the promotion of the agile mindset is important [35].

As Papadopoulos [49] mentioned, the adoption of an agile framework can lead to an improvement in quality, flexibility, and employee as well as customer satisfaction, in comparison to development teams continuously working according to traditional development methods. Furthermore, he recognized that the time and energy consuming adoption, especially in large organizations, is difficult [49]. Because with an increase in organizational size an increase in complexity follows [23]. Svensson and Höst [63] revealed the complexity of the initial phase of agile development. The side effects beyond the involved developer are immense to management and stakeholders [63]. Direct support by management simplifies transformations, however is not always provided. Consequently, the involvement of the management must be promoted [63] to take their fears and benefit from their support [36]. Also, all other stakeholders should be convinced to support the agile transformation [18].

Without being able to reform the mindset agile transformations do not succeed and this mindset change requires enough time. On the other hand, during an adoption, the enthusiasm decreases if the developers experience no success for a long time. Consequently, it is important to find the right balance in time management [35]. To probe the environment pi-

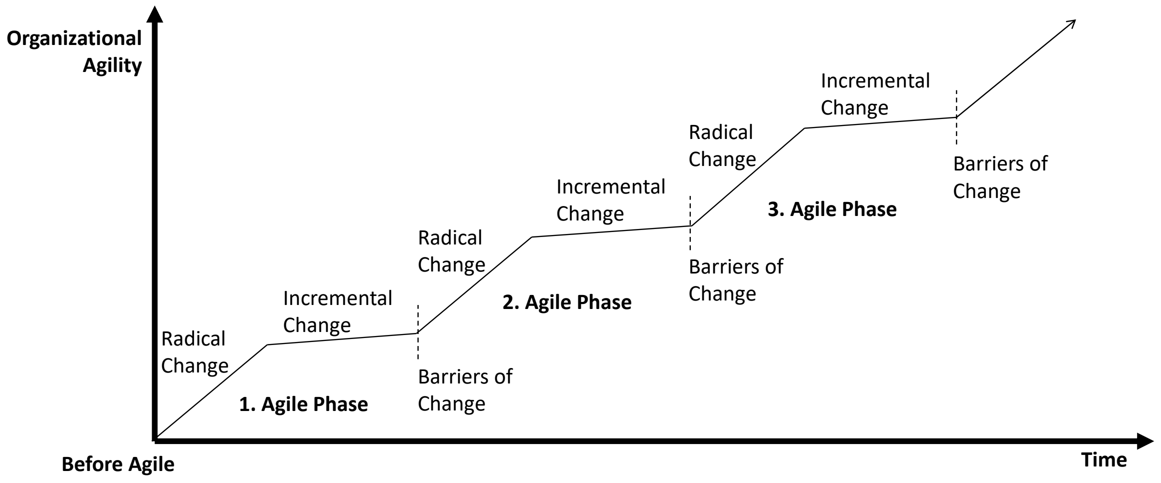

Figure 2.3.: "Large-Scale Agile Transformation Process as Episodic Change" by Fuchs and Hess [27]

loting can save cost and time, while gaining experience [18,35]. As clarified previously, the mindset change at all levels facilitates the transformation [18, 24, 35]. Such as every transformation, an agile one is a change in daily habits of employees and can lead to misunderstandings and skepticism. If all relevant parties are involved, this can be avoided [18, 35] and enterprise-wide transparency can be ensured [68].

Fuchs and Hess [27] clustered the adoption in different phases to be able to compare them. Figure 2.3 presents the evolving of agility in the organization over the phases. But a transformation can be done in different ways. Elssamadisy [25] offered a step by step pattern language to iterative adopt agile practices. 10 years later Kalenda et al. [35] recognized successful transformations do not need a concrete scheme but work out better individually. Moreover, Kalenda et al. [35] and Fuchs and Hess [27] proposed to tailor the transformation process to get a customized adoption for one's own organization.

3. Related Work

After explaining the relevant main terms in the last chapter, this chapter continues with an overview about related work regarding transformations in large-scale agile development. Especially, scaling practices, concerns, and success factors reported in previous research are compared. Case studies as well as literature reviews were done. The following insights in eight different articles are presented chronological from 2015 until now. To sum up, Table 3.2 gives a short overview about the entire discussed related work at the end of this chapter.

Papadopoulos (2015): Moving from Traditional to Agile Software Development Methodologies also on Large, Distributed Projects [49]

Papdoupoulos [49] conducted a case study to analyze the transformation process of a software and service company. Similar to our research, a large organization was observed, but unlike our case organization, the software and service company was geographically distributed [49]. Following, some challenges he recognized are not assumed to find in our study.

The following scaling practices are recommended by said scientist [49]:

1. "Multi-team backlog" - a shared backlog can lead to more transparency, while a separated backlog for every team can increase the focus on the tasks.
2. "Multiple meetings" - individual team meetings for each team facilitate conduction of these meetings, while meetings with participants from several teams provide more information.
3. "Scaling the Infrastructure" - the environment, such as tooling, must be scaled.
4. "Organizational agility" - the whole organization should live the agile mindset.

In the end, Papadopoulos [49] recognized an improvement in quality and employee satisfaction, as well as an increased flexibility when urgent requests and content changes appeared. Furthermore, the study showed that the transformation in a large, distributed organization needed more planning than in smaller ones [49].

Paasivaara and Lassenius (2016): Scaling Scrum in a Large Globally Distributed Organization: A Case Study [48]

Paasivaara and Lassenius [48] performed a case study at a large global telecommunication company. They observed the scaling from two to 20 agile development teams, distributed across four sites [48]. The case organization applied LeSS, this differs from the organization examined by us, besides our case organization is not distributed. Paasivaara and Lassenius [48] interviewed different roles during the empirical data collection to distinguish role-specific impressions and concerns [48].

The adopted scaling practices described were [48]:

1. Adopting a scaling framework: LeSS - The framework offered various scaling practices, which were tested partly.
2. Common meetings - Shared common sprint planning, retrospective, demo, and scrum-of-scrums meeting with all teams were done.
3. Scaled Product Owner - An Area Product Owner role additional to the product owner role was responsible for an entire area together with some development teams (which still had a regular product owner, too).

In summary, Paasivaara and Lassenius [48] recognized a performing agile organization. Unfortunately, LeSS limited the opportunities of the complex organization and under time pressure some agile practices were not used according to the framework. LeSS may work better if the product splitting is clearer and product areas have lower inter-dependencies [48].

Dikert et al. (2016): Challenges and success factors for large-scale agile transformations: A systematic literature review [18]

In 2016 Dikert et al. [18] performed a systematic literature review in which especially enterprise transformations have been considered. Through this study, they recognized 35 challenges and 29 success factors for large-scale agile transformations. Both, afterwards, have been categorized, into nine and 11 categories, to get a sorted view [18].

The nine categories which cluster their found challenges sorted by their frequency (from often to less often) are [18]:

1. "Agile difficult to implement"
2. "Integrating non-development functions"
3. "Change resistance"
4. "Requirements engineering challenges"
5. "Hierarchical management and organizational boundaries"
6. "Lack of investment"

7. "Coordination challenges in multi-team environment"
8. "Different approaches emerge in a multi-team environment"
9. "Quality assurance challenges"

The success factors have already been shown in Section 2.3.3. Most frequently found were: "Choosing and customizing the agile approach [,] Management support [,] and Mindset and Alignment" [18].

Kalenda et al. (2018): Scaling agile in large organizations: Practice, challenges, and success factors [35]

Kalenda et al. [35] conducted a literature review and case study at a software company in 2018 to determine practices, challenges, and success factors in large-scale agile development.

The main scaling practices identified were [35]:
1. Scaled meetings - Scaled retrospectives, scaled plannings, scaled sprint demos/reviews, and Scrum of Scrums meetings helped to scale agile development in an organization.
2. Undone department - Specific expert teams from this department supported the development teams with their expertise to solve undone tasks.
3. Communities of Practice - Analogical to guilds, the Community of Practice was a network meeting frequently and discussing specific topics (see also Section 2.3.3 and Pattern Candidate **CO-01:** COMMUNITY OF PRACTICE in Section 5.2.2 for more details).
4. Scaling of requirements management - To clarify requirements about tasks solved by more than one development team further self organized meetings and tools were added.

Kalenda et al. [35] recognized 10 different challenges in total, nine of them through literature review and one more during their case study. Three of the nine challenges found in their literature review also appeared during their case study (see Table 3.1). As already mentioned in Section 2.3.3, Kalenda et al. [35] additional identified several success factors through literature review and demonstrated two of them in their case study, while finding two previously unpublished success factors.

Challenges by Kalenda et al. [35]	**Literature Review**	**Case Study**
Resistance to change	✓	✓
Distributed Environment	✓	
Quality assurance issues	✓	✓
Integration with non-agile parts of organization	✓	✓
Lack of commitment and teamwork	✓	
Too much pressure and workload	✓	
Lack of knowledge, coaching and training	✓	
Requirements management hierarchy	✓	
Measuring progress	✓	
Too fast roll-out		✓

Table 3.1.: Comparison between challenges found by Kalenda et al. [35] in their literature review and their case study

Uludağ et al. (2018): Identifying and Structuring Challenges in Large-Scale Agile Development Based on a Structured Literature Review [67]

Uludağ et al. [67] aimed to create a new pattern language including concerns, stakeholders, and patterns. The Large-Scale Agile Development Pattern Graph (see Figure 3.1) is published online [65] and gives an overview about all related information. They conducted a structured literature review, also using the work of Dikert et al. [18] and found several stakeholders and challenges. Recognized stakeholders for their pattern language are Development Team, Product Owner, Scrum Master, Software Architect, Test Team, Product Manager, Program Manager, Agile Coach, Enterprise Architect, Business Analyst, Solution Architect, Portfolio Manager, Support Engineer, and UX Expert [67]. Identified challenges were structured into following categories [67]:

- "culture [and] mindset,
- communication [and] coordination,
- enterprise architecture,
- geographical distribution,
- knowledge management,
- methodology,
- project management,
- quality assurance,
- requirements engineering,
- software architecture, and
- tooling"

to sort and analyze them. For every reported challenge related stakeholders, category, sources, and an ID were also documented. Some challenges were not related to specific stakeholders, but were program specific [67].

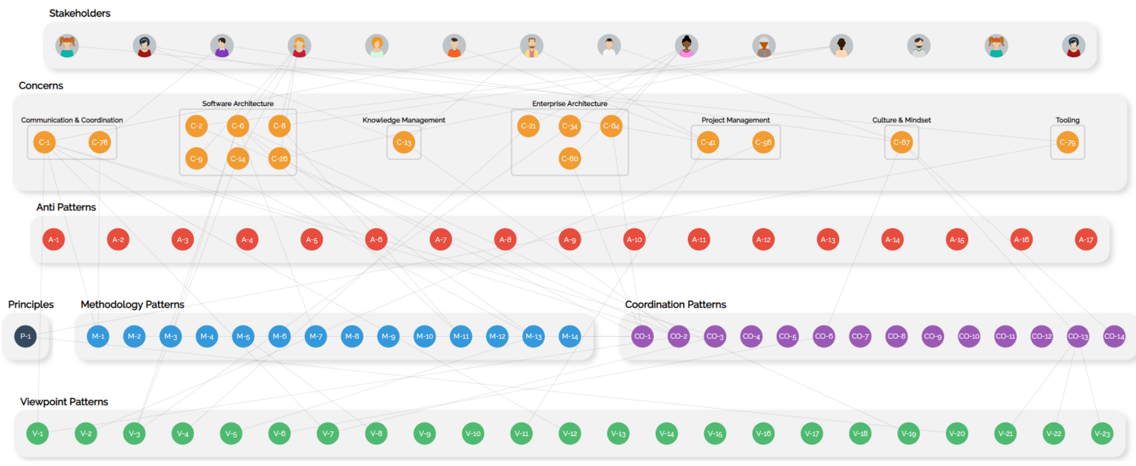

Figure 3.1.: Large-Scale Agile Development Pattern Graph by Uludağ et al. [65]

The study done by Uludağ et al. [67] was just theoretical, so a validation of the founded challenges in practice was missing. Several scientists [30, 66, 68] conducted research based on some of their challenges and demonstrated a selection of these by investigating agile coaches and scrum masters in large-scale agile development through empirical studies.

Fuchs and Hess (2018): Becoming Agile in the Digital Transformation: The Process of a Large-Scale Agile Transformation [27]

Fuchs and Hess [27] compared the agile phases of two case organizations during their transformation and analyzed every phase those organizations went through. Both organizations were not IT companies even though, following a large-scale agile adoption. Fuchs and Hess [27] choose four phases every firm passes through: "before agile, 1. agile phase, 2. agile phase [,] and 3. agile phase" [27]. The first phase for example represented the piloting, whereas the second agile phase illustrated the restructuring of the organization (changes influencing the management of the developers) as well as further piloting (see Figure 2.3 in Chapter 2). The phases were used to compare the two case organizations but did not represented special milestones. They compared the actions done through the phases and the evolution of the challenges plus the barriers, and found analogies as well as differences. A gaining in agility from phase to phase was noticed [27]. Recurring barrier in all phases was the "Coordination of different organizational worlds" [27]. Additionally, the selection of workers was experienced as concern as well as the individual adaption of agile methods.

Uludağ et al. (2019): Documenting Recurring Concerns and Patterns in Large-Scale Agile Development [66]

To inspect the practical necessity of their pattern language (cf. [67]) Uludağ et al. [66] conducted interviews with experts from different organizations regarding large-scale agile development. Most experts had more than three years agile experience. The interviewees were requested to rate the practical usability of the components of their pattern language, resulting the pattern language was rate as useful [66].

Additionally, concerns and patterns were analyzed through interviews. Defined to exemplify for the large-scale agile development pattern language, these four pattern were explained in detail by Uludağ et al. [66]:

- **P-1:** STRICTLY SEPERATE BUILD AND RUN STAGES
- **CO-1:** COMMUNITY OF PRACTICES FOR ARCHITECTURE
- **V-1:** ITERATION DEPENDENCY MATRIX
- **A-13:** GOLDEN HAMMER

Uludağ and Matthes (2019): Identifying and Documenting Recurring Concerns and Best Practices of Agile Coaches and Scrum Masters in Large-Scale Agile Development [70]

With this research, Uludağ and Matthes [70] aimed to evaluate and extend their pattern language (cf. [65,66]). During interviews with scrum masters as well as agile coaches several concerns were identified and analyzed. They recognized, that the adaption of frameworks must be conducted careful. One of their Anti-Patterns is "Don't use scaling agile frameworks as a recipe" and demonstrates how controversial and complex the adaption of scaling frameworks can be [70]. Through the investigation of patterns contrary opinions were found. Their patterns should be applied individual instead of used as cooking recipe. Overall, agile coaches as well as scrum masters faced several concerns during their daily work life. Especially the demonstration of "the value add of agile methods" [70] as well as other concerns regarding the mindset and coaching of their colleagues came up.

Considering all mentioned related literature, we conducted our study and had the previous findings in mind to recognize and investigate their use at the automotive company we observed. As illustrated in Table 3.2, both, literature as well as empirical research, was done before with various focuses.

Author	Publication date	Type	Focus
Papadopoulos [49]	2015	case study	practices and challenges especially during the adoption
Paasivaara and Lassenius [48]	2016	case study	scaling practices
Dikert, Paasivaara and Lassenius [18]	2016	literature review	35 challenges and 29 success factors
Kalenda, Hyna and Rossi [35]	2018	literature review followed by a case study	several practices, 9 success factors and 10 challenges
Uludağ, Kleehaus, Capano and Matthes [67]	2018	literature review	14 stakeholder roles and 79 challenges (for their pattern language)
Fuchs and Hess [27]	2018	case study	evolution of challenges and barriers through several phases of the transformation
Uludağ, Harders and Matthes [66]	2019	empirical research	concerns and example patterns which extend their pattern language
Uludağ and Matthes [70]	2019	pattern-based design research	concerns and pattern (related to agile coaches and scrum masters) which extend their pattern language

Table 3.2.: Overview about related work

4. Case Study

This chapter gives an overview about the case study design, including an explanation of data collection techniques (shadowing, feedback talks, interviews, and provided data) used for this empirical research. Later on, this chapter describes the case structure and its environment, including an overview of the investigated automotive organization. Finally, insights in the transformation conducted at the organization are provided and found scaling practices are presented.

4.1. Case Study Design

The data collection was based on the taxonomy according to Lethbridge et al. [40]. As mentioned previously in Section 1.3, the described work clustered data collection techniques in three degrees, as displayed in Table 4.1. The data collection for this thesis was primarily done with first and third degree data collection techniques. The following techniques were used to study the development work, to analyze the structure of the organization and to identify recurring concerns and good practices:

First degree: Observations of (mainly agile) meetings and workshops (see Section 4.1.1), unstructured interviews (so-called feedback talks with members of development teams and other stakeholders; see Section 4.1.2) and semi-structured interviews with people affected by agile transformation (see Section 4.1.3) were conducted.

Third degree: Reports of prior intern analysis, tables with structural, organizational, and technical information, wiki pages of the development teams about their work, slides, and coaching material (see Section 4.1.4).

Category	**Statement**
First Degree	direct involvement of software engineers
Second Degree	indirect involvement of software engineers
Third Degree	study of work artifacts only

Table 4.1.: Data collection techniques suitable for field studies according to Lethbridge et al. [40]

The mentioned data collection techniques are demonstrated in the following four sections.

4.1.1. Shadowing

Lethbridge et al. [40] explained shadowing as a process where the researcher attends the objects. For this thesis shadowing was used to get an overview about the organization, the structure of the teams, and to be part of the agile work at the OEM. The observation started simultaneously with the transformation. First goal was to observe the transformation process and to get an overview about agile development work. Consequently, pilot projects and their agile meetings were observed and reported. Daily scrum meetings, sprint plannings, sprint reviews, sprint retrospectives, and some backlog refinements were visited about a time of five months. We also followed teams to workshops and events by different constellations (e.g. events by **CO-01:** COMMUNITY OF PRACTICE see more in Section 5.2). Most observations took place in pilot teams. This is because those teams were able to compare their current agility to their own evolution and act as trendsetter in the department. All visited meetings were recorded afterwards and coded to ascertain recurring concerns and used good practices.

4.1.2. Feedback Talks

Feedback talks were executed as informal and unstructured interviews on a frequent foundation (cf. [53]) with different stakeholders. Especially, conversations with members of the team for mentoring and implementing agile work methods were conducted. Therefore, we got a broad overview about their experience in the development teams, with affected managers, and stakeholders. The discussed topics changed individually by the actual issues and matters of the interviewees. Most feedback talks started with a short exchange about current activities and new practices. Furthermore, often benefits and concerns of agile work methods were broached. As the feedback talks were informal and unstructured no static process was followed. Even the duration varied between ten to sixty minutes. Three rounds of feedback talks have been conducted. Between the second and third round semi-structured interviews as explained in the next section took place.

4.1.3. Interviews

Semi-structured interviews, according to Runeson and Höst [55], have been implemented to get a general overview of the personal insights of different roles [40]. Our semi-structured interviews contained open questions as well as closed ones [55]. The questionnaires can be found in the Appendix A. All interviews lasted between 60 and 80 minutes, except of one with the head of the department, because of scheduling reasons. Longer interviews were not feasible, because of tight schedules of interviewees. The solution was to conduct 60 minutes interviews and stay as close as possible to the questionnaire, if applicable the interviews took up to 80 minutes. All interviews were taken at the site of the organization in German language face-to-face. The order of the questions varied slightly from interview to interview. All interviews were recorded meanwhile and coded afterwards.

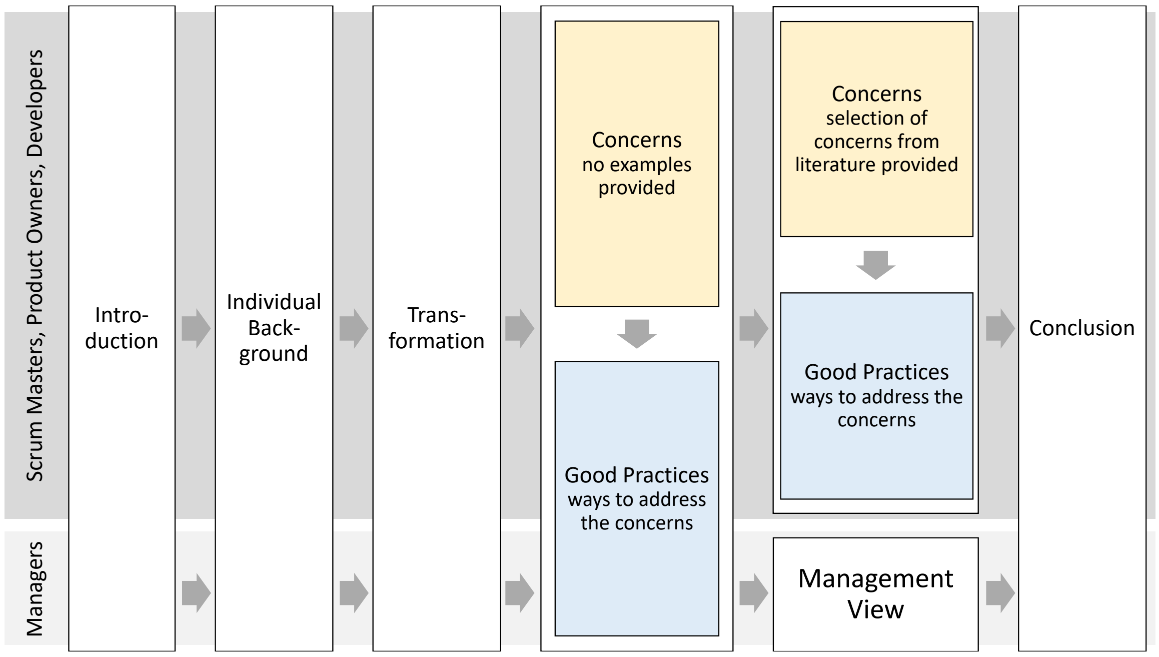
Scrum Masters, Product Owners, Developers
Managers
Intro-duction
Individual Back-ground
Trans-formation
Concerns no examples provided
Good Practices ways to address the concerns
Concerns selection of concerns from literature provided
Good Practices ways to address the concerns
Management View
Conclusion

Figure 4.1.: Process of the conducted semi-structured interviews

The process of the interviews for the four roles scrum master, product owner, developer, and manager is presented in Figure 4.1. Each Interview started with an *introduction* to explain the importance of the interview, the goal of this case study, and clarifying the conditions of the interview. Afterwards a set of questions about the *individual background* of the interviewee and his team followed. A few questions regarding the *transformation* and its realization followed. The main part of the interviews was about concerns experienced by the interviewee and used practices to address these. First the interview partner was asked to list some recurring *concerns* he faced while adopting agile development and if already done, how these concerns were addressed by using *good practices*. No inspiration or examples were offered by the researcher. The second part of our interviews differed according to the different roles: Product owners, scrum masters, and developers continued talking about *concerns*. We provided a list with concerns from literature, especially a selection of recurring concerns listed by Uludağ et al. [67] and clustered into different categories. They were asked how often a concern appeared and additionally, if already done, how they addressed this with *good practices*. The selection is explained later in this section in detail. The possible solutions explained by the interviewees to solve arising concerns were analyzed and documented as pattern candidates. The second half of the main part varied for mangers: Instead of showing them the selection of concerns found by Uludağ et al. [67] they were asked more intensively about their *management view*, including the realization of the transformation, role specific concerns, and success stories. All interviewees were requested to suggest practices for other professionals in the same role. As a *conclusion* the advantages and disadvantages of the agile work methods were questioned and the interviewees were asked, whether they would like to continue agile development or prefer to return to other development methods.

Uludağ et al. [67] listed 79 challenges based on a grounded literature review. Adapted from the feedback talks done before and the observation, 56 concerns out of the 79 have been selected and were presented in the interviews. Some of the 79 concerns were obviously not relevant for this study, for example off-shore concerns, which took not place, because the department was not distributed. The rest was selected based on previous experiences at the case organization. Shadowing as well as insights gained by feedback talks were used to identify possible concerns for the interview partner. The categories have been: communication and coordination, culture and mindset, enterprise architecture, methodology, knowledge management, geographic distribution (regarding room situations), project management, requirements engineering and software architecture. The concerns varied in some aspects for the product owners, scrum masters, and developers.

The leaders were asked different questions in the second part of the main body, because the aim was to get deeper insights in the goals of the transformation, the structure of the organization, the main practices provided by the management, the project story, and different management levels (cf. [48]).

The choice of the interview partners was done random through the observation of agile meetings. This ensured a broad range of roles and experience levels and guaranteed diversity. The participation in the interviews was voluntary for all interview partners. Finally 14 interviews have been conducted with five different roles: product owner, scrum master, developer, and manager (team leader and head of the department), an overview of the roles can be found in Table 4.2.

Role	Amount of Interviewees
Product Owner	2
Scrum Master	4
Developer	5
Team Leader	2
Head of Department	1
Sum	14

Table 4.2.: Interviewee overview

The interviewees were picked out of six teams in the department, to get a comparison of the arising concerns and used practices between the results of different roles in the same team. The detailed relation between interviewees, their role, and their team is shown in Table 4.3. The head of the department is not assigned to a special team and consequently not listed in this table. Two scrum masters, two managers, and two product owners were responsible for two development teams each.

Team	Scrum Master	Product Owner	Developer	Team Leader
A	1	1	1	1
B			2	
C	1	1		1
D				
E	1		2	
F	1			
Sum	4	2	5	2

Table 4.3.: Distribution of interviewees into teams

4.1.4. Provided Data

All information provided by the case organization for the use of this case study was used to get a more detailed perspective and was handled as third degree data collection by Lethbridge et al. [40]. Especially material (slides and coaching material) for the implementation of the transformation and workshop material about the vision of the department and main department was included. Also, material about other departments working agile in the company and their good practices were taken into account. The organization itself even provided reports of analysis. Moreover, tables which consisted of structural, organizational, and technical information by the OEM were useful. The wiki pages of the development teams facilitated the shadowing of their meetings and the understanding of their work.

After the case study design, next the description of the OEM and the conducted transformation follows. First information about the organization and the department for vehicle dynamics development are given. Further, process, motivation, and piloting of the transformation are described. Later on, used scaling practices are demonstrated.

4.2. Case Description

This case study took place at a large German OEM in the vehicle dynamics development. The automotive company is internationally aligned and has more than 100.000 employees in around 15 countries. Last year, they sold more than 2.5 million vehicles and generated a revenue about approximately 100 billion euro. The monitored department is part of a main department for vehicle dynamics development with around 1.000 employees and located at one site in different buildings. The department itself contains around 140 developers, mainly producing software elements and architecture models in different roles: function developer, software developer, function security agent, function architect, software architect, diagnostic analyst, and others. Some also have project management duties. They only develop for intern stakeholders, for the advanced development and the serial development of the vehicle. Both are influenced by high regulatory and legislative requirements and standards. Their long term goal is a 100% scaled agile vehicle dynamics development, starting with the transformation explained in the next paragraphs.

Transformation

One and a half year before the observation, first steps towards the transformation from a traditional matrix organization to agile development started (see Figure 4.2). In the traditional matrix organization conventional role-oriented teams worked together according to the v-model (cf. [7]). The teams were structured skill-oriented. Interchange, when needed, was driven by upcoming topics. To get complexity under control, the OEM started a frequent exchange in project circles and optimized roles as well as tasks for a better cooperation. Additionally, roles and tasks were optimized. The next step was to gain flexibility with agile development. First pilot projects started about one year before the observation. Each pilot was established in different circumstances as occurring. All started to use agile practices and adopted the agile mindset. They aimed to loose the structure of roles and tasks. The transformation for the entire department to work agile started approximately one year later, same time the observation began. Cross-functional teams were established to maximize flexibility. With the transformation of the department a restructuring of the main department for vehicle dynamics development took place. This was because of different organizational reasons and additional had the goal to prepare the main department for the future.

The motivation for the transformation was felt different by different colleagues. As explained before in Section 4.1.3 all interview partners were asked what they perceived as motivation. For half of the interview partners flexibility was a motivator to work agile. The second most important reason for the transformation was to manage changing priorities in the future. More than a quarter of all interviewees mentioned the reduction of time-to-market and project risks, a better matching with their partner departments, which already worked agile, and improvements in their team moral as motivating. Last was

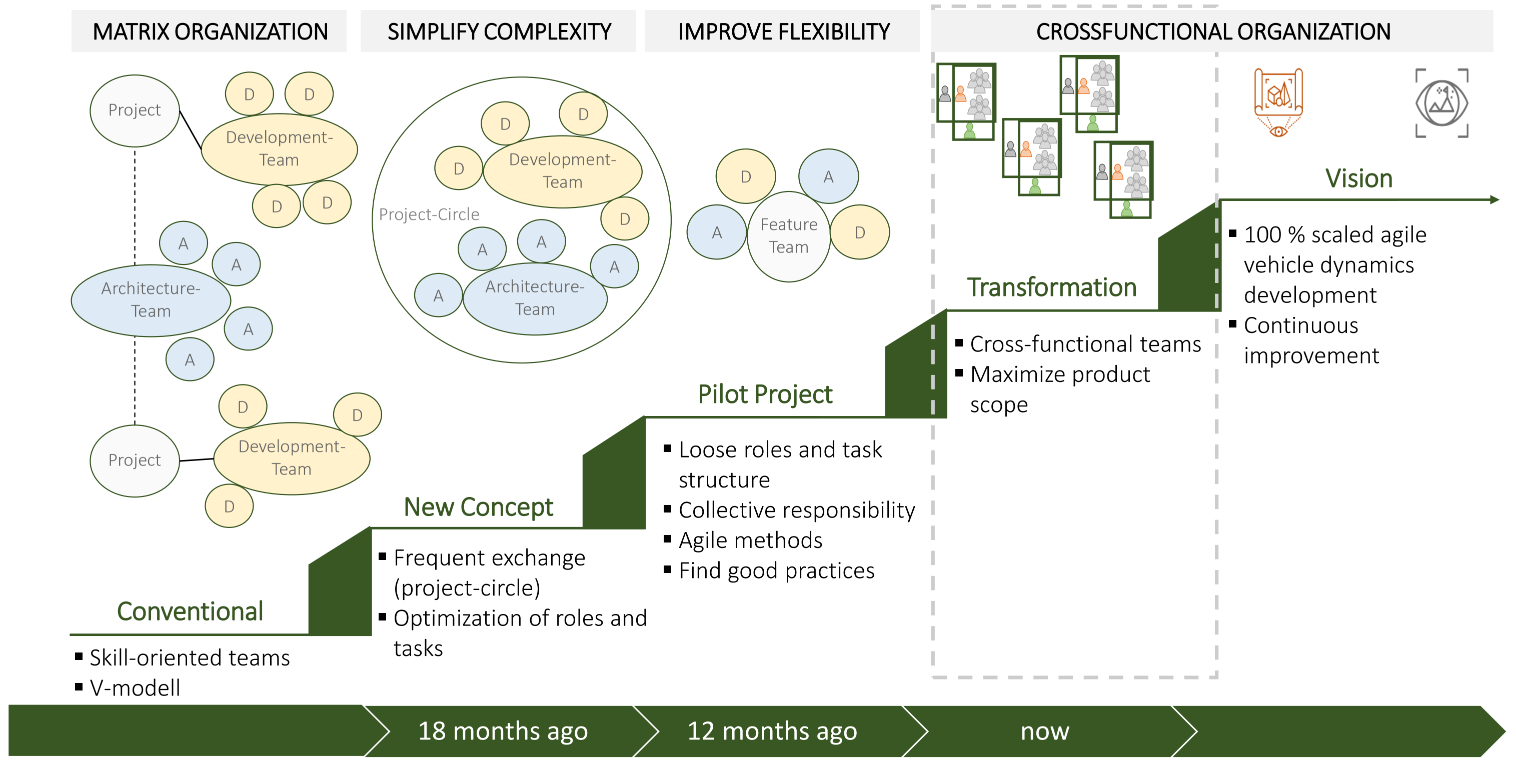

Figure 4.2.: Transformation from a traditional matrix organization to a cross-functional organization

never mentioned by development team members, but one leader, two product owners, and one scrum master stated it. Figure 4.3 shows an overview about all stated transformation reasons. Colored in blue are the possible motivators suggested to all interviewees and colored in orange are reasons which were stated proactively by them. Three out of four scrum masters felt misunderstood by higher management. They portrayed some managers thought agile development results in higher efficiency and faster development. This was confirmed by a higher manager during our research. One manager also addressed the trend for agile development may influence the department as well as the company-wide rising demand for agility.

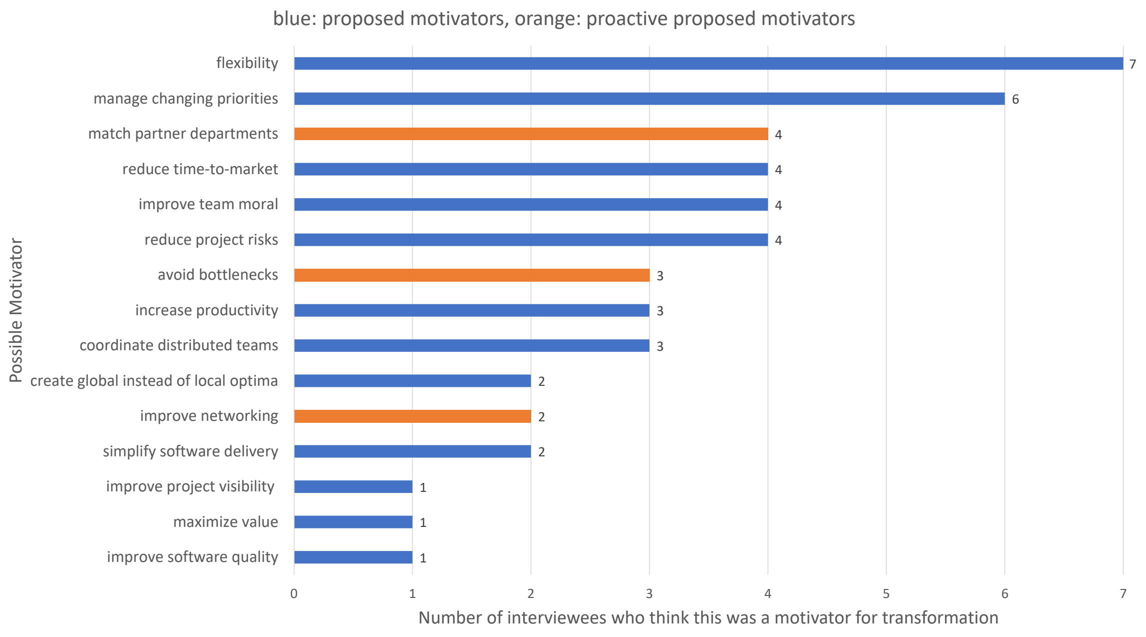

Figure 4.3.: Overview of motivators for the transformation mentioned by interviewees

As clarified before, the transformation started with several pilot projects: Seven teams already started three to 14 months before the actual transformation, as illustrated in Figure 4.4. As this figure shows, one leader has disciplinary responsibility for one or two feature teams and one product owner. The leader has no disciplinary management to the scrum master. The scrum master guarantees a trouble-free agile development. He escalates impediments and is the main contact regarding agile work for the team, the product owner and the leader. The product owner is responsible for all technical content, while the team and the product owner can seek for advise by the leader, but the leader should not proactively influence how the development team solves its tasks or how the product owner prioritizes and organizes their backlog. As in servant leadership, the manager ful-

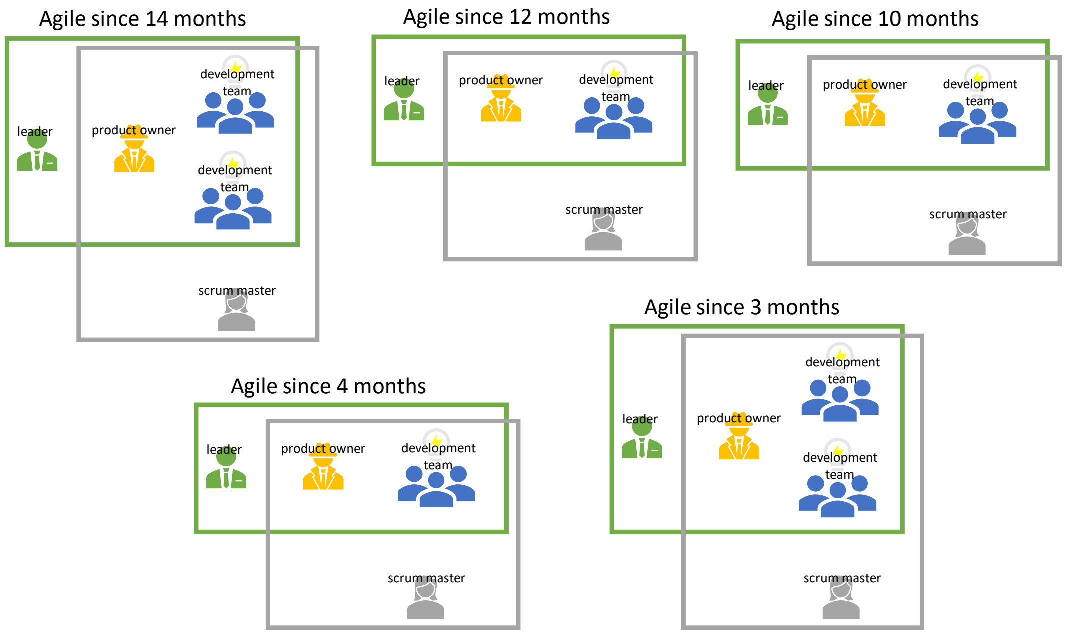

Figure 4.4.: Overview about pilot development teams (including the duration of their piloting)

fills two roles: the role of the disciplinary supervisor as well as the servant, who helps the developers to work without impediments and assists when needed [29]. Ordinarily development teams contain between six and eight developers. One exceptional case was team E which contained 10 developers directly involved in the agile development and four external developers still working according to the v-model because of their compliance. With the agile work they tried to loose their strict roles and work as T-shaped developers. This means, that they are experts in a specific field and have a broad and general knowledge outside their core [45]. This led to a growth of knowledge in all development teams. Each team was cross-functional responsible for one feature. For this reason they decided to call themselves "feature team" instead of "development team". The majority of development teams knew each other before and have worked together for some years in the traditional organization, not organized in one team but in the same department. In summary, first cross-functional teams in the department were pilot teams. Since the beginning of the regular transformation additional teams planed and implemented their cross-functional structure.

Most of the pilot projects organized themselves with the help of professional external and internal scrum masters independent from the other pilots. Only four of the seven have been scaled together. Divided in two groups of two development teams they shared a backlog, product owner, scrum master, and their meetings. Some had experiences with agile development before, but most developers did not. They learned through own workshops, done by their scrum masters to explain the agile mindset and wording, professional training, shadowing of other agile teams (see pattern candidate **M-06:** SHADOWING in Appendix B.1.5), online literature, and learning by doing. During the interviews developers, who started without the support of a professional scrum master, mentioned this as a possible fail factor. During the initial state of the agile development the assistance by a scrum master was experienced as necessary and helpful.

The development teams in the pilot have been authorized to choose their own preferred agile framework. This was interpreted differently. In most teams developers decided self-organized to use Scrum, nevertheless one manager forced his team to use it. In the end, all teams oriented their agile development to Scrum, even though Scrum was initially designed for small, individual, and co-located teams (cf. [48]) and is not easy adoptable to large organizations. In the beginning this was no concern because all development teams still worked individually. During the transformation the management of the department offered every team the same freedom of choice to pick their preferred framework. Most chose Scrum again.

Scaling Agile Development at the Case Organization

The scaling of the agile work at the case organization was performed by three main factors: a dedicated work unit regarding agile methodology, the COMMUNITY OF PRACTICE, and additionally the EMPOWERED COMMUNITY OF PRACTICE (also see their pattern candidates **CO-01:** COMMUNITY OF PRACTICE and **M-01:** EMPOWERED COMMUNITY OF PRACTICE in Chapter 5), and a Product Owner Board. Further other practices were used, but these three were considered as most important. Identified good practices are explained in the next chapter.

To supervise the agile work a new work unit responsible for the implementation and mentoring of agile work methods was installed. This team of scrum masters, agile coaches, and development specialists led the case organization through the agile transformation. As mentioned before, the scrum masters were not assigned to the managers of the development teams. They were assigned to this separate work unit to avoid disciplinary pressure by managers and to facilitate the assigning of scrum masters to development teams. In addition, the work unit was experienced as a good way for networking between scrum masters. For networking, they organized additional meetings with scrum masters from other departments at the company to discuss concerns and exchange practices. This work unit provided guidance for development teams, not only in the initial state, and offered workshops lasting several hours up to several days for the different demands of development teams. They also guaranteed to answer requests by the management regarding agility and the agile coaching of the management. Additionally, the installation of tools to assist agile development and wiki platforms was supported. This work unit dedicated agile work methods is not only a change team (cf. [70]), but also responsible for the agile development in the future after the transition.

Within the beginning of the transformation, the first COMMUNITY OF PRACTICE (see Pattern Candidate **CO-01**) and their extension, the EMPOWERED COMMUNITY OF PRACTICE (see Pattern Candidate **M-01**), were set up. As Paasivaara and Lassenius [47] explained in their paper about "communities of practice in large distributed agile software development organization", the COMMUNITY OF PRACTICE in agile development differ from other known communities. They need a domain, community, and practice to evolve. The domain is the topic which connects the community. For example in our case organization one established COMMUNITY OF PRACTICE was assigned to architecture. The community of a COMMUNITY OF PRACTICE includes all interested developers from different development teams. Their practices varied. Most have been observed to share knowledge as well as to discuss current concerns and interesting cases regarding their domain. Additionally, they generated processes and tools and discussed universal topics. Paasivaara and Lassenius [47] investigated eight characteristics of a successful COMMUNITY OF PRACTICE [47]. To compare these characteristics with the adoption of the COMMUNITY OF PRACTICE and the EMPOWERED COMMUNITY OF PRACTICE in the case organization Table 4.4 illustrates the differences as well as similarities at the end of this chapter. A COMMUNITY OF PRACTICE can end as fast as it started, it lasts only as long as the community participates [47]. As only the first months of the transformation have been observed, no COMMUNITY OF PRACTICE declined but many new evolved.

As said before, also their extensions (**M-01:** EMPOWERED COMMUNITY OF PRACTICE) started. They had additional rights compared to the usual COMMUNITY OF PRACTICE. While a COMMUNITY OF PRACTICE was not obligatory but voluntary and was not automatically allowed to make decisions valid for all development teams, the EMPOWERED COMMUNITY OF PRACTICE was both. That means, an EMPOWERED COMMUNITY OF PRACTICE had the right to make decisions which had to be respected by all development teams. Additionally from every relevant development team one developer must be sent to the EMPOWERED COMMUNITY OF PRACTICE. In contrast to a usual COMMUNITY OF PRACTICE the EMPOWERED COMMUNITY OF PRACTICE was approved and established by the management. The management chose which COMMUNITY OF PRACTICE must be empowered by using the agile decision making model provided by Uludağ et al. [71] and decided for which development teams the content was relevant and following obligatory. As shown with exemplary content in Figure 4.5 the decision-making model contains two axis: the x-axis represents the scope, which means how many feature teams will be influenced by a topic. The y-axis represents intern regulations and costs. If a topic is low in one or both a COMMUNITY OF PRACTICE but no EMPOWERED COMMUNITY OF PRACTICE is needed. For example artificial intelligence (AI) was used by some teams but was rated low in costs and regulations. Consequently, a COMMUNITY OF PRACTICE was created, whereas

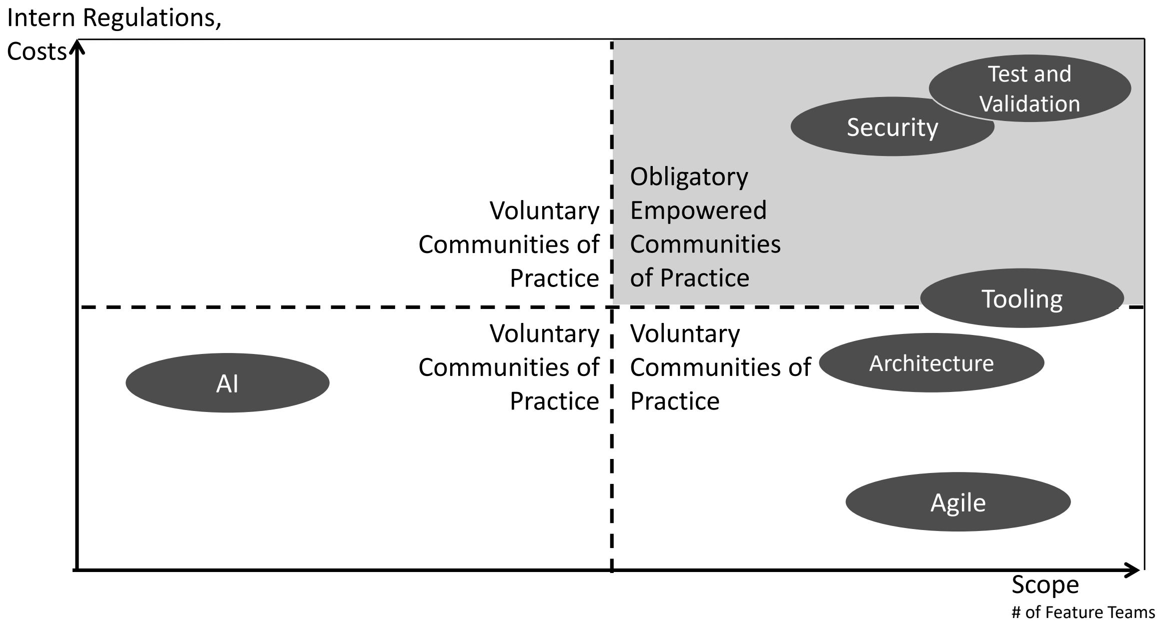
Intern Regulations, Costs
Test and Validation
Security
Voluntary Communities of Practice
Obligatory Empowered Communities of Practice
Tooling
Voluntary Communities of Practice
Voluntary Communities of Practice
Architecture
AI
Agile
Scope
# of Feature Teams

Figure 4.5.: Agile Decision-Making Model by Uludağ et al. [71] for the COMMUNITY OF PRACTICE and the EMPOWERED COMMUNITY OF PRACTICE

for security an EMPOWERED COMMUNITY OF PRACTICE was established. This domain is high in both, because intern regulations are strict to achieve the regulations by the legislator and relevant for all development teams producing software and models for the vehicle.

Another method used for scaling is the Product Owner Board. This is a circle where the product owners and the higher management of the department and main department came together frequently to create and prioritize additional tasks for an additional shared backlog. All stakeholders were allowed to bring tasks into the circle if they rated them as highly important. Consequently, the higher management decided about it. The Product Owner Board was also used for the exchange of the product owners. They exchanged tasks and coordinated them to the best fitting feature team. This led to a networking advantage for them. The board also coordinated the communication with other departments in the company.

Characteristics by Paasivaara and Lassenius [47]	**Adaption of Communities of Practice** (see **CO-01:** COMMUNITY OF PRACTICE)	**Adaption of Empowered Communities of Practice** (see **M-01:** EMPOWERED COMMUNITY OF PRACTICE)
Interesting topic and concrete benefits	Participants were involved in the decision making process, were invited to bring own suggestions for topics and used the chance to stay updated about upcoming topics in the domain of the Community of Practice. Furthermore, they gained knowledge as well as competencies.	All arguments of the COMMUNITY OF PRACTICE are valid too. But as the participation was obligatory it was possible that not each participant was interested in the topic.
Passionate leader	As the selection of the leader was done by the participants, they selected another leader in case the leader was not able to motivate everyone.	The management selected a leader.
Proper agenda	Previously the agenda was sent to all possible participants, everyone was invited to add own items to the agenda.	
Decision making authority	Provided with the needed authority the COMMUNITY OF PRACTICE evolved processes and tools as long as the decisions did not influence other domains.	They had the authority to make decisions valid for everyone. If decisions influenced another EMPOWERED COMMUNITY OF PRACTICE, they were requested to collaborate.
Open community	Every interested colleague was invited to participate.	
Supporting tools to create transparency	Wiki pages about every COMMUNITY OF PRACTICE and each EMPOWERED COMMUNITY OF PRACTICE were established, but transparency about the decision making process and discussions was just guaranteed for participants.	
Suitable rhythm	The participants decided how often they needed to met.	
Cross-site participation when needed	No cross-site participation was needed because the department was located at one site.	

Table 4.4.: Comparison between characteristics of successful Communities of Practice [47] and the adaption at the case organization

5. Observing and Identifying Recurring Concerns and Good Practices

This chapter summarizes all relevant results based on the data collection at the case organization. First, the recurring concerns which have been mentioned by the interviewees and identified through shadowing are clarified in Chapter 5.1. They are separated in the ones which have been based on literature and the ones mentioned proactively. Good practices have been investigated at the case organization, were edited as pattern candidates, and are presented in the second part of this chapter (see 5.2). Because good practices have been identified at just our case organization they are pattern candidates, not pattern.

5.1. Recurring Concerns

During the rigorous shadowing of the agile development at the case organization (see Chapter 4.1.1), the unstructured feedback talks with several different stakeholders and employees in different situations of the transformation (see Chapter 4.1.2), and the conducted semi-structured interviews (see Chapter 4.1.3) with 14 employees of the department various concerns have been identified. The findings extend the list of 79 concerns by Uludağ et al. [67] which was previously expanded by Harders [30] to in total 115 concerns. During the second part of the conducted interviews, a selection of 56 concerns of the 79 concerns by Uludağ et al. have been presented to the interviewees. These had the chance to annotate and label which concerns came up during the agile development. Since the observation took place during the transformation phase not all concerns appeared the whole time, some had been more important at the beginning, some not before the pilot teams had a few months experience (see Figure 5.1).

As the interviewees were first asked to proactively mention their concerns, they stated some of the concerns which also were listed in the selection of the concerns prepared for the second part of the interviews. The duplicates were merged directly during the evaluation and coding of the interviews. Only concerns which were mentioned in the first phase of the interview and were also in the list of concerns for the second part were counted as duplicates. The merging guaranteed to have a well-arranged list of concerns. 55 of the selected and presented 56 concerns were stated as recurring at the case organization. Additional 29 new concerns were found during the interviews. The new findings are listed by their descending order, regarding their frequency. The numbering starts with 116, because 115 concerns (79 by Uludağ et al. [67], and 36 more by Harders [30]) were already identified:

C-116: ***How to collaborate with the traditional world?*** The clash of traditional and agile development leads to concerns which influence not only the agile working people but also the ones still working according to the traditional work methods. The interaction with the remaining organization is not clear for everyone, since there are different dependencies and different contacts. The satisfaction of all parties throughout the hierarchy is hard to gain. Both must cooperate to find compromises, e.g. when communicating shorter cycles to facilitate the working in sprints for the agile development and still ensuring a smooth working for the other party with minimal additional expenses. Supplementary, explaining the agile work (and getting rid of false rumors) to the traditional working people and giving the highest possible transparency has to be done. Related to this concern is the "[p]roblematic coordination with other business units" [27], which was also recognized by Fuchs and Hess in their case study. Also Kalenda et al. recognized the "[i]ntegration with nonagile part of organization" as recurring concern found in literature they analyzed as well as in the organization they observed [35].

C-117: ***How to handle false rumors?*** In management forums the agile development is sometimes seen as a more efficient and faster working method. It is hard to get rid of such rumors. From several viewpoints it is not clear when exactly such claims first emerged. They do not lead to the right agile mindset.
Ulludağ and Matthes' [70] "**C-91:** *How to demonstrate the value add of agile methods?*" as Harders' [30] "**C-113:** *[How to create] the [a]gile [m]indset for [h]igher [m]anagement[?]*" refer to this concern. The claim can be avoided by understanding the motivation and the real improvement of agile development, as well as creating the real agile mindset through all management levels. A scrum master mentioned that false rumours came up more often in higher management.

C-118: ***How to handle regulatory standards?*** High regulatory legal standards and company-standards can complicate the work. Some regulatory standards are in contrast to the agile mindset. For example, standards require specific documentations, while the Agile Manifesto prefers "working software over comprehensive documentation" [74]. Regulatory standards equate both.

C-119: ***How to coordinate imperfect room situations?*** No open project areas, rooms for open discussions, and spontaneous meetings impede developers from working together in one place.

C-120: ***How to guarantee the right understanding of roles?*** All persons concerned must understand the new roles scrum master and product owner. Additionally, they need to understand the evolution of their own role, especially leaders, who should only

act as a disciplinary manager and should give the developer the needed freedom to work. Sometimes the leader must solve tasks which were in the traditional development assigned to the project manager, but are not assigned to the product owner in the agile work, so bottlenecks accrue. Also developers must understand their evolution and their growing responsibility. The right filling of positions gains relevance for every participant, because of the closer cooperation. During working the fulfillment of all roles must be guaranteed. This leads to concerns if the roles are not understood.
This concern is related to **C-58:** *How to deal "with loss of management control"?* by Uludağ et al. [67]. These scientists mentioned exactly the change of the management role and the missing understanding for this change, as well as **C-99:** *How to handle "missing understanding of roles"?* by Harders [30] describes. She describes with this concern the understanding of the roles meaning, while the new concern we found describes the understanding of the evolution of roles and the working following the new responsibility.

C-121: ***How to reduce sideline activities?*** Developers need to solve urgent tasks which come up spontaneously and cannot be solved by others. Because of this they cannot spend their entire time for the sprint tasks. Because nobody else is responsible for the sideline tasks, they needed to solve them - otherwise nobody would solve them. Scrum masters and developers feel powerless and do not know how to avoid these sideline activities.

C-122: ***How to find time for training?*** Teams (and their manager) must take their time to learn and must be willing to spend time for sharing knowledge instead of using it for developing.

C-123: ***How to simplify a complex tooling environment?*** The tooling environment creates local instead of global optima, which also happened before, but gains relevance through agile work. The tooling environment must be customized and facilitated for the agile development, which takes time to develop. So every developer can simply use the tooling environment, without a long training period. If given tools are to complicated, one developer mentioned that the decision to work without tools and use own solutions requires courage and should be made in the team.

C-124: ***How to share knowledge in the team to back-up failures?*** Agile work benefits from the chance to back-up the work of others. T-shaped people give developers the possibility to choose the part of work they enjoy the most. But on the other hand, this requires discipline and perseverance while also training aspects they might not enjoy.

C-125: ***How to distribute competencies across the team?*** The set up of competencies across the team requires time, patience, and the aspiration of every single developer to educate oneself further.

C-126: ***How to find the same wording for agile terms?*** The same word does not automatically describe the same meaning for everyone. Especially inexperienced agile participants do not use the right wording and often do not understand what is meant by others. The language barrier slows done the developing and the common vision.

C-127: ***How to recognize the need for self-discipline?*** Agile developing requires a lot more self-discipline by every single developer. Without this discipline the sprint cannot be fulfilled as planned. The organization of the personal work life, which was more individual before, influences the entire team.
This concern is related to "**C-95:** *How to deal with lacking orientation due to missing leadership?*" by Uludağ and Matthes [70] which describes the decreasing degree of leadership through agile development and the following personal responsibility of developers as part of their team.

The following three concerns have been mentioned by only two, not the needed three times, of the interviewees proactively through the first phase of the interview. Three independent times are needed to call the concerns recurring (cf. [16]). Consequently, the following three concerns are not automatically recurring. Nevertheless they are, because they have also been observed in other empirical studies by Harders [30], Dikert et al. [18], Fuchs and Hess [27] or Kalenda et al. [35].

C-128: ***How to work without the needed support by the management?*** If the management, especially the first row leaders, does not support the development teams, the agile developing cannot evolve. The management must live the agile mindset as well as the operating developers have to. Additionally, the managers are the leading partners for the developers to avoid frustration.
This concern is related to Harders' **C-113:** *How to create "the [a]gile [m]indset for [h]igher [m]anagement"?* [30]. The establishment of an agile mindset in the management supports the development teams. Also Dikert et al. [18] mentioned that the unwillingness of the management to change occurred in 10 percent of their observed organizations. Additionally, they recognized the "Management in waterfall mode" [18] as a concern in 14 percent of their observed organizations, which was also monitored at the case organization. Also Fuchs and Hess [27] recognized a "[l]ack of top management engagement" during the agile transformation in their case study.

C-129: ***How to prevent old wine in new bottles?*** A new name does not automatically bring new forms of developing. All participants must take care that "no old wine in new bottles" (cf. [41]) is served. One product owner mentioned that arose especially in the initial phase of agile working.
The concern "Reverting to the old way of working" by Dikert et al. [18] is related to this one. Especially in stress situations and during the transformation agile teams need to take special care to stay in the agile mindset. Equally Kalenda et al. [35] recognized that quality assurance issues by using old instead of agile methods have been found in the literature they analyzed as well as in the case they observed.

C-130: ***How to avoid a growing pressure to single developers due to the growing responsibility for the team?*** Fulfillment of the given roles and a shared responsibility are needed. Otherwise, single developers experience a growing pressure due to the feeling of being responsible for the whole team, if the team does not share the responsibility. The pressure for developers was also recognized as a recurring concern through the literature analyzed by Kalenda et al. [35].

The following concerns have been mentioned by one or two interviewees independently. As a result, these are not recurring regarding to Coplien [16]. However, they have been declared as concerns which the interviewees had to face.

C-131: ***How to ensure networking and knowledge sharing across several development teams?*** The initial phase of communities was experienced as a concern: The right casting of the position community coordinator (also called community lead), the motivation of developers, and the guarantee that no knowledge gets lost needed to be ensured.

C-132: ***How to handle long waiting times due to dependencies?*** Long cycles between the start of architecture and its software release because of dependencies and testing phases hinder releasing potentially shippable product increment at the end of a sprint.

C-133: ***How to handle separate development (advanced development and serial development)?*** The difference between advanced development and serial development in a sprint is insignificant, anyways it is significant for single developers. This leads to differences. Urgent requests for serial development are often cost intensive, so they should be handled first.

C-134: ***How to prioritize complex and intricate items?*** The product owner has problems to prioritize the complex and intricate backlog items without the help of the development teams. This makes the product owner to be a bottleneck and requires time by the team members. Also a very large backlog causes concerns, while the backlog is not clearly shown, so prioritizing is hard.

C-135: ***How to avoid knowledge isles?*** Punctual knowledge is spread over the whole company, but only saved in the minds and nowhere documented. Knowledge isles complicate the sharing of knowledge and increase the risk to loose knowledge when working cross-functional instead of skill-oriented.

C-136: ***How to prevent justifications for being the pilot project?*** Pilots worry, that they need to justify their new way of working and explain everyone their motivation as well as their improvements regarding traditional way of working. This was experienced as an emotional strain and costs time.

C-137: ***How to escalate impediments?*** Neutral and rational phrasing of impediments must be learned. Also the acceptance for different opinions and an open minded position in discussions to escalate impediments. This is not only relevant for the development team members (as one developer mentioned), but also for the communication between manager, product owner, scrum master, and the development team. Also a lack of technical understanding of externals should not be felt as irritating.

C-138: ***How to facilitate refinements?*** Without realizing the importance of refinement meetings development team members do not understand them as an improvement for their sprint but instead experienced refinement meetings as a waste.

C-139: ***How to establish a team harmony?*** Harmony in teams now has a higher value than it had before. The leader and the scrum master must take the responsibility to take care of a harmonious cooperation.

C-140: ***How to handle lone fighters?*** Some individualists need more time until they think and act as team worker. They need more attention by the scrum master, product owner, and leader. It takes also patience by all other team participants to involve them in the team.

C-141: ***How to apply test driven development without testing in the own team?*** For test driven development the test concept must be located (partly) in the own development team. If this is not the case writing tests twice results in a waste of code. Additional bugs can be the result of different test cases.

C-142: ***How to unite developers from different organizational units?*** Developers from different organizational units who work together in one development team experience political pressure and inequality. This leads to a negative dynamic in the team. Especially, if this concern and **C-121:** *How to reduce sideline activities?* come together, political stress results.

These concerns did not came up all at the same time. As the interviewees mentioned and the analysis of the observation showed some happened in the earlier phase of the piloting while others gained importance later. Also the duration varied. The arising of the recurring concerns is shown in Figure 5.1, especially the increasing and decreasing of the concerns. The time frame is clustered in agile phases as also Fuchs and Hess [27] used. The first agile phase represents the piloting, while the second one represents the phase restructuring the department. In our case organization the adoption of scaled agile development started in April 2019 (cf. start of the second agile phase in Figure 5.1).
C-119: *How to coordinate imperfect room situations?* is not listed, because this concern was addressed by creating more project rooms and changing the seating arrangement in the office, consequently it is not related to special agile phases.

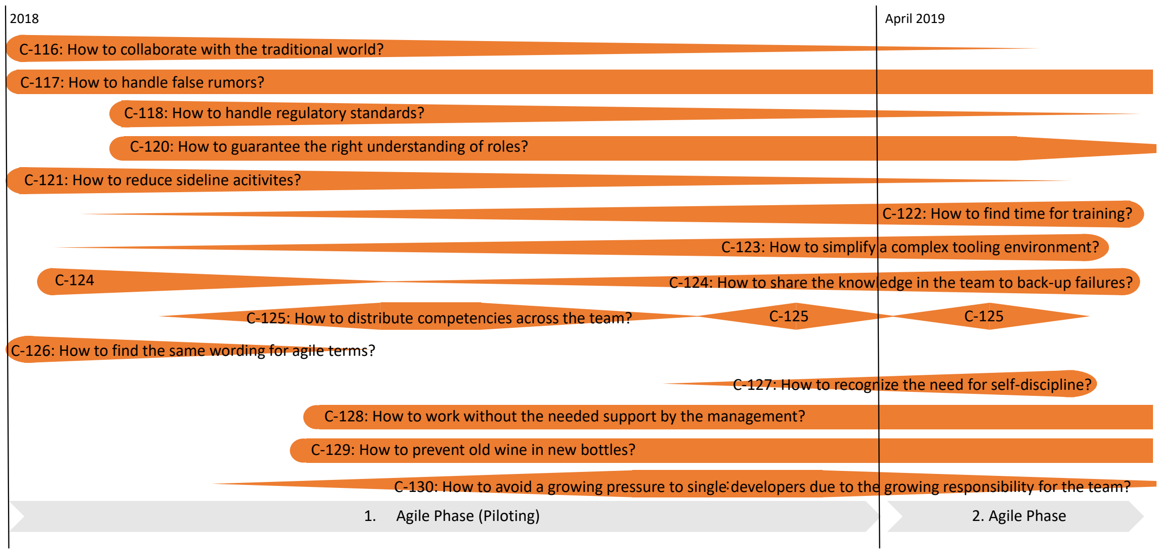

Figure 5.1.: Chronological sequence of the occurrence of the new recurring concerns

All concerns are assigned to categories. The scheme for the categories was first introduced by Uludağ et al. in their listing of 79 concerns [67]. The following categories are used:

1. Culture and Mindset,
2. Knowledge Management,
3. Methodology,
4. Communication and Coordination,
5. Requirements Engineering,
6. Project Management,
7. Tooling,
8. Software Architecture and
9. Quality Assurance.

Table 5.1 offers an overview of all new concerns and these categories, while Figure 5.2 shows the distribution of their categories. For reasons of comparability, the categories-distribution of all found concerns is illustrated in Figure 5.3.

ID	Concern	Category
C-116	*How to collaborate with the traditional world?*	Culture and Mindset
C-117	*How to handle false rumors?*	Culture and Mindset
C-118	*How to handle regulatory standards?*	Requirements Engineering
C-119	*How to coordinate imperfect room situations?*	Communication and Coordination
C-120	*How to guarantee the right understanding of roles?*	Methodology
C-121	*How to reduce sideline activities?*	Project Management
C-122	*How to find time for training?*	Knowledge Management
C-123	*How to simplify a complex tooling environment?*	Tooling
C-124	*How to share the knowledge in the team to back-up failures?*	Knowledge Management
C-125	*How to distribute competencies across the team?*	Knowledge Management
C-126	*How to find the same wording for agile terms?*	Methodology
C-127	*How to recognize the need for self-discipline?*	Culture and Mindset
C-128	*How to work without the needed support by the management?*	Culture and Mindset
C-129	*How to prevent old wine in new bottles?*	Culture and Mindset
C-130	*How to avoid a growing pressure to single developers due to the growing responsibility for the team?*	Culture and Mindset
C-131	*How to ensure networking and knowledge sharing across several development teams?*	Methodology
C-132	*How to handle long waiting times due to dependencies?*	Software Architecture
C-133	*How to handle seperate development (advanced development and serial development)?*	Project Management
C-134	*How to prioritize complex and intricate items?*	Requirements Engineering
C-135	*How to avoid knowledge isles?*	Knowledge Management
C-136	*How to prevent justifications for being the pilot project?*	Culture and Mindset
C-137	*How to escalate impediments?*	Communication and Coordination
C-138	*How to facilitate refinements?*	Methodology
C-139	*How to establish a team harmony?*	Culture and Mindset
C-140	*How to handle lone fighters?*	Culture and Mindset
C-141	*How to apply test driven development without testing in the own team?*	Quality Assurance
C-142	*How to unite developers from different organizational units in one development team?*	Communication and Coordination

Table 5.1.: New findings on concerns and their categories

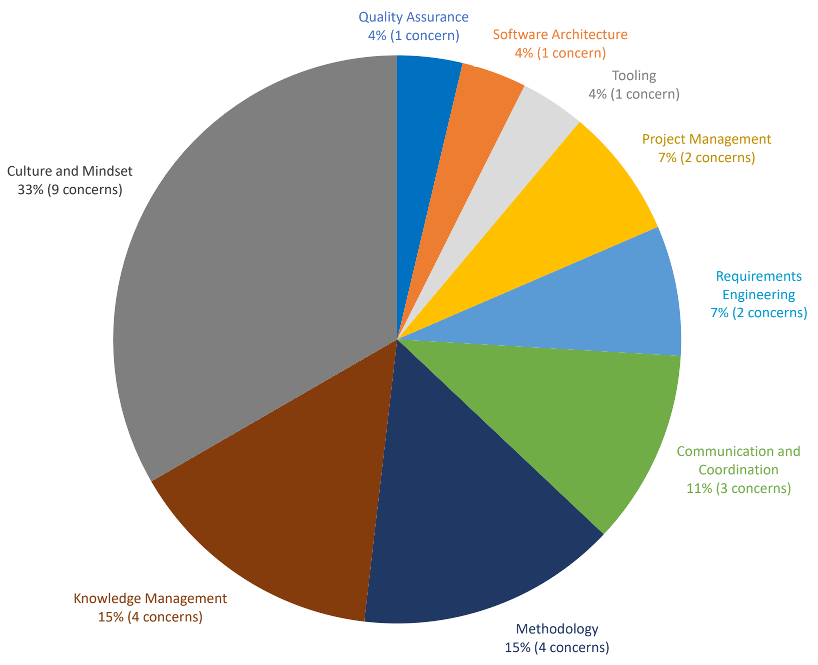

Figure 5.2.: Distribution of categories of new found concerns

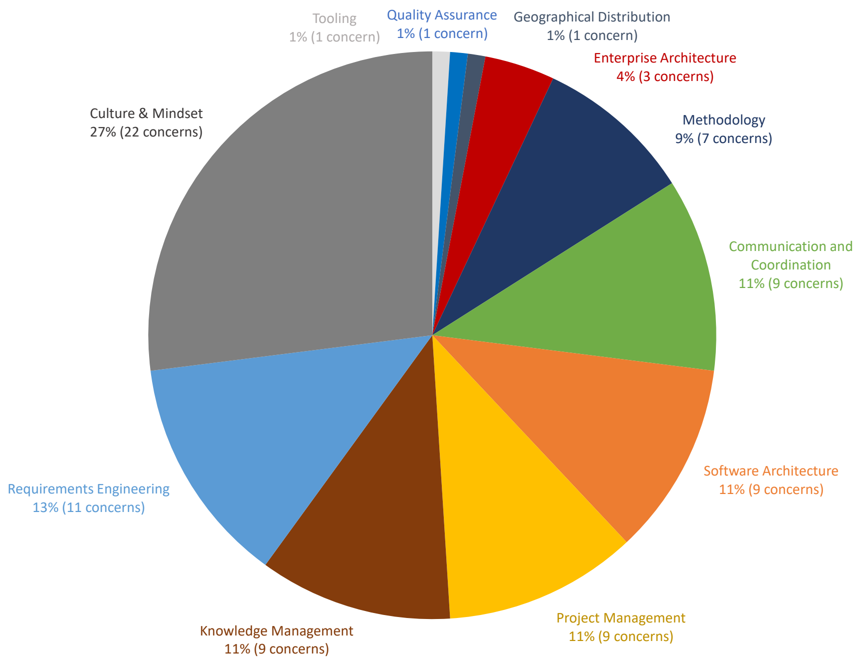

Figure 5.3.: Distribution of categories of all found concerns

Figure 5.4 and Figure 5.5 present all concerns which occurred in the case organization, excluding duplicates because these have been merged before. To see which role picked which concerns, Figure 5.4 shows how often a scrum master, product owner, developer or manager mentioned proactively one of the concerns during the first phase of the interview. The interviewees did not know which concerns were mentioned by others. Figure 5.5 shows the same comparison for the selection of concerns which were presented to the interviewee. Since this part of the interview was specially conducted with scrum masters, product owners, and developers only, these roles are shown in the graph.

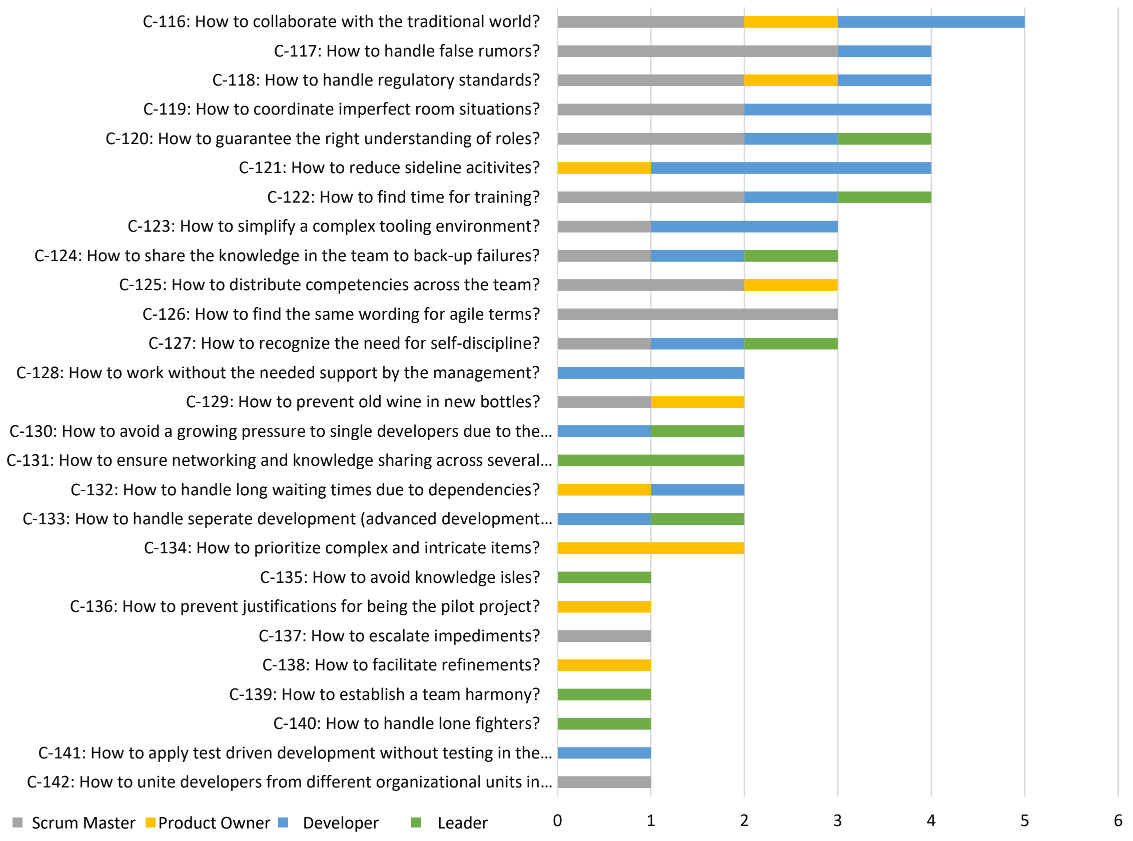

Figure 5.4.: Overview about which concerns were mentioned proactively how often by the four different roles (scrum master, product owner, developer, and leader)

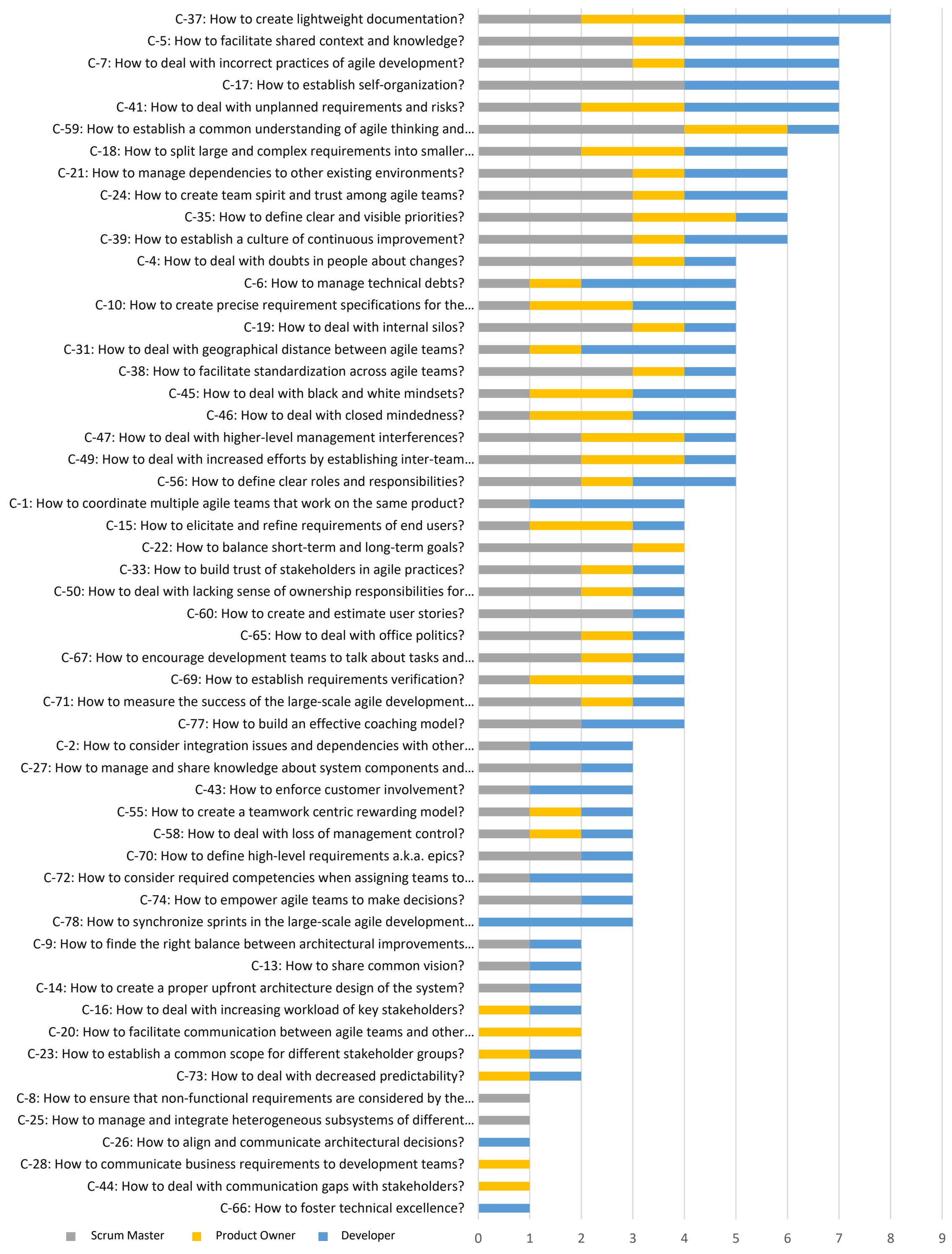

Figure 5.5.: Overview about which concerns of the selection were annotated how often by scrum masters, product owners, and developers

To sum up, 12 new recurring concerns have been found through the analysis of agile development during the transformation at the case organization. Three more concerns have been identified which also can be classified as recurring in regards to the literature where they appeared too. The interviewees had to face 12 more new concerns, which were not classified as recurring. 55 concerns from the presented selection were labeled as occurring at the case organization.

The concerns vary in their complexity. For example **C-119:** *How to coordinate imperfect room situations?* was solved at the case organization by improving the room situation. New project rooms and open spaces were established for feature teams. Another example to mention is **C-132:** *How to handle long waiting time due to dependencies?*: Feature teams from the pilot project started to coordinate their suppliers delivery period with their sprint times. New suppliers guaranteed to develop agile, used the same sprint rhythm, and the product owners coordinated the sprint backlog to avoid waiting times. However, some concerns seemed to be solved easier than others. Especially concerns regarding the category *Culture and Mindset* took more time, as several interviewees pointed out. Since the organization aimed to eliminate these too, they adopted several good practices which we explain in the next chapter.

5.2. Good Practices

At the case organization several good practices have been identified. These were analyzed and summarized as pattern candidates. 17 pattern candidates have been found. One Principle, two Coordination Patterns, seven Methodology Patterns, six Viewpoint Patterns, and one Anti-Pattern. An overview of all found pattern is given in Table 5.2.

ID	Pattern Candidate
	Principle
P-01	CREATE T-SHAPED PEOPLE
	Coordination Patterns
CO-01	COMMUNITY OF PRACTICE
CO-02	BACKLOG GROOMING
	Methodology Patterns
M-01	EMPOWERED COMMUNITY OF PRACTICE
M-02	PILOTING
M-03	TRAVELLING
M-04	SHARE THE CHANGE
M-05	COME TO OUR DEMOS
M-06	SHADOWING
M-07	SHARE A MAILBOX
	Viewpoint Patterns
V-01	RADAR CHART
V-02	ROADMAP
V-03	PROJECT PLAN
V-04	SYNCHRONIZED CALENDAR
V-05	STARFISH
V-06	BURNDOWN CHART
	Anti-Pattern
A-01	DON'T COMBINE DEVELOPERS FROM DIFFERENT ORGANIZATIONAL UNITS IN ONE DEVELOPMENT TEAM

Table 5.2.: Pattern candidates identified at the case organization

The relationships between observed roles, pattern candidates itself, and occurring concerns which they address are graphically presented in Figure 5.6. As not all practices were established in the same time, Figure 5.7 demonstrates the process of main scaling practices (special work unit for agile methods, COMMUNITY OF PRACTICE with EMPOWERED COMMUNITY OF PRACTICE, and Product Owner Board), as well as all Methodology Pattern Candidates and Viewpoint Pattern Candidates which can be assigned to a special time.

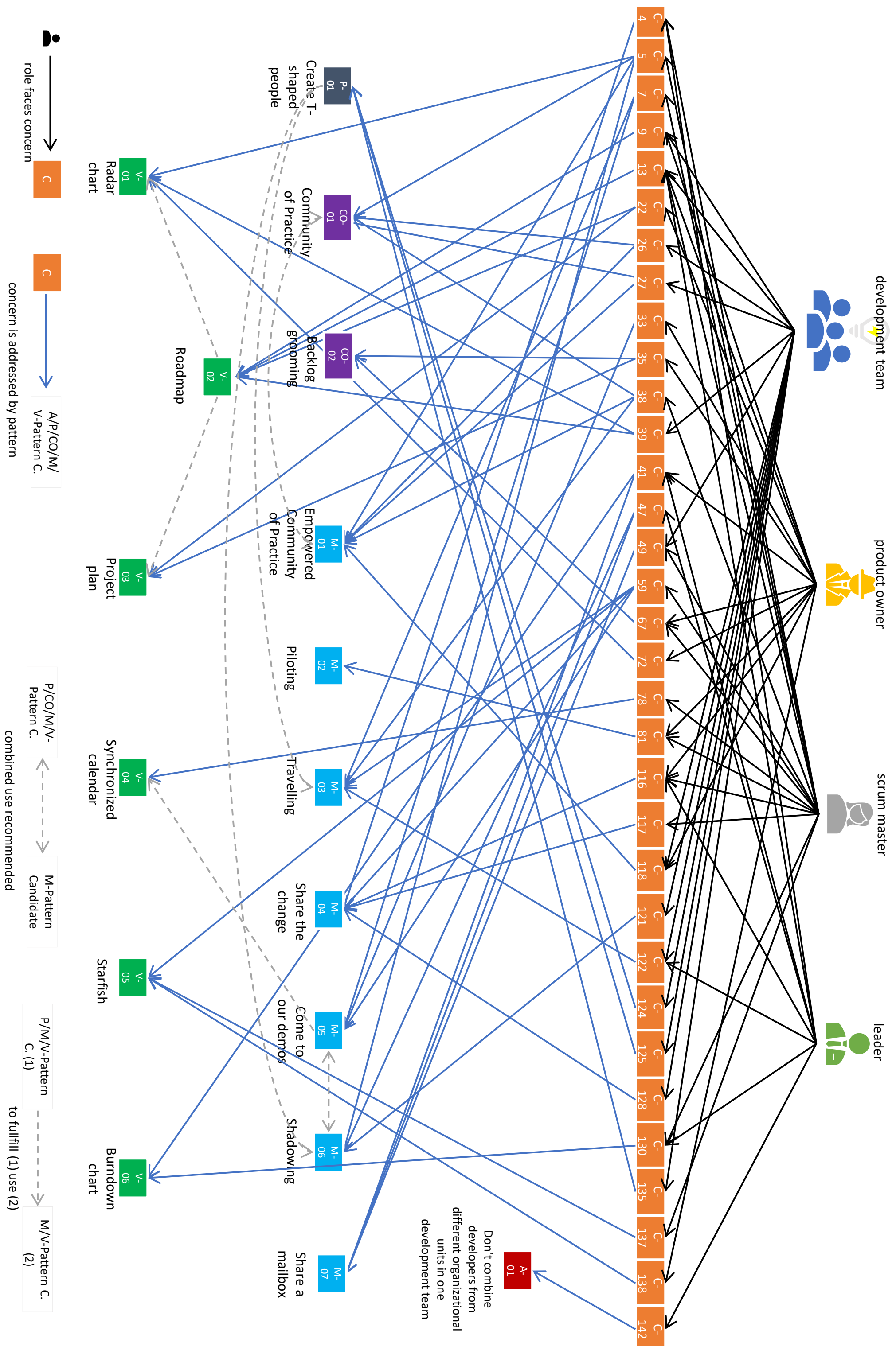

Figure 5.6.: Relationship between different roles, their concerns, and the identified pattern candidates

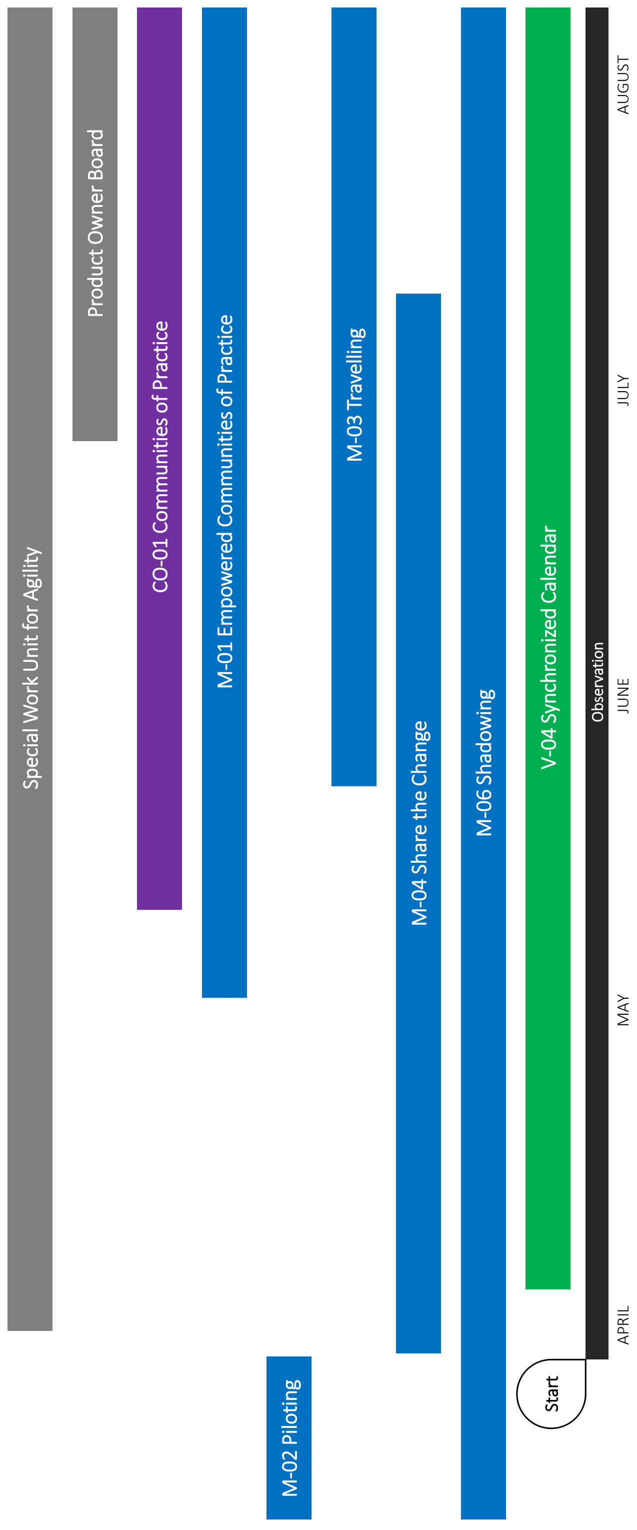

Figure 5.7.: Process of the establishment and use of scaling practices and selected pattern candidates

In the following sections one Principle Candidate (**P-01:** CREATE T-SHAPED PEOPLE), one Coordination Pattern Candidate (**CO-01:** COMMUNITY OF PRACTICE), two Methodology Pattern Candidates (**M-01:** EMPOWERED COMMUNITY OF PRACTICE and **M-03:** TRAVELLING), one Viewpoint Pattern Candidate (**V-04:** SYNCHRONIZED CALENDAR), and one Anti-Pattern Candidate (**A-01:** DON'T COMBINE DEVELOPERS FROM DIFFERENT ORGANIZATIONAL UNITS IN ONE DEVELOPMENT TEAM) are explained in detail. The explanation of all other pattern candidates can be found in Appendix B.

5.2.1. Create T-Shaped People

Pattern Overview	
ID	P-01
Name	CREATE T-SHAPED PEOPLE
Alias	-
Summary	T-shaped professionals are generalists as well as specialists [45]. With them interdisciplinary cooperation can be evolved. This is why in feature - but not skill-oriented teams, t-shaped people are needed.
Type	Communication
Binding Nature	Recommended

Example

In one team several competencies are distributed, but colleagues cannot backup each other because they do not understand the work of the others.

Context

In agile development a new, broad distribution of knowledge can lead to more success (cf. [35]).

Problem

- **C-124:** *How to share the knowledge in the team to back-up failures?*
- **C-125:** *How to distribute competencies across the team?*
- **C-135:** *How to avoid knowledge isles?*

Forces

- General as well as specialist knowledge requires the motivation to be taken by the employees and is challenging, especially for long-time experienced experts as they do not recognize the need for broad knowledge.

Solution

Train developers to be both, generalists in their subject as well as experts in their specific domain. That requires open-minded developers and the motivation for training sessions.

Consequences

Benefits:

- Team members should be able to share tasks domain-independently in their team and be able to support each other, if bottlenecks occur.
- Increases the possibilities to substitute colleagues, while they are not available.
- Higher motivation due to various possible tasks.

Liabilities:

- Requires training for employees and costs capacity (investment costs).

See also

M-03: TRAVELLING
M-06: SHADOWING

5.2.2. Community of Practice

Pattern Overview	
ID	CO-01
Name	COMMUNITY OF PRACTICE
Alias	Community
Summary	A COMMUNITY OF PRACTICE is an informal network between development teams regarding a special domain. The domain can be technical, agile, or other.

Example

Several colleagues from different teams complain about a missing network.

Context

Agile working employees may seek for interaction and discussion with like-minded people (cf. [47]). Furthermore, knowledge management across different development teams must be guaranteed. Knowledge sharing in feature-oriented development teams differs from the knowledge sharing in skill-oriented development teams.

Problem

- **C-5:** *How to facilitate shared context and knowledge?*
- **C-26:** *How to align and communicate architectural decisions?*
- **C-27:** *How to manage and share knowledge about system components and their dependencies?*
- **C-38:** *How to facilitate standardization across agile teams?*

Forces

- Colleagues must be motivated to work in a COMMUNITY OF PRACTICE.
- If the participants do not use the COMMUNITY OF PRACTICE, or not enough people participate at all, the COMMUNITY OF PRACTICE makes less sense.

Solution

A Community of Practice can be founded by any interested person regarding a specific topic. Everyone can be part of it, the participation is voluntary. Usually the members of a Community of Practice are from different development teams. A Community of Practice is no team and consequently does not need to work with an own backlog. However, they can establish their own backlog or use any other tools, if needed. Tasks are solved directly together or handed over to development teams if they require more time or are too complex. The participants meet frequently and can have different kinds of meetings: workshops, work meetings, stand ups, pitches, and more. Also informal meetings off the job are possible, especially when the domain is not directly regarding the job. The Community of Practice-Coordinator organizes the meetings.

A Community of Practice works well for knowledge-management and networking across feature teams. Nevertheless, a Community of Practice is not authorized to reach decisions automatically effective for all development teams (cf. [47]).
Main responsibility of the Community of Practice is the knowledge as well as competency management, the enhancement of processes and tools, and the discussion of overarching topics. With their work they support the feature teams with new methods, tools, and processes. As Paasivaara and Lassenius [47] analyzed, a Community of Practice can be used for continuous improvements.

Consequences

Benefits:

- Given foundation for networking.
- Higher transparency because topics are discussed in the open Community of Practice, instead of closed circles and everyone is invited to join.
- Facilitates knowledge management, because all interested and experienced colleagues come together.
- Solves problems with the help of others.
- Broad knowledge base and memory for everyone.

Liabilities:

- Time intensive because of additional meetings.
  Possible solution: Save time in every sprint for the participation. At the case organization, 10 percent of their sprint time was saved.

See also

M-01 Empowered Community of Practice

5.2.3. Empowered Community of Practice

Pattern Overview	
ID	M-01
Name	EMPOWERED COMMUNITY OF PRACTICE
Alias	Core Community of Practice
Summary	Additional to the characteristics of a COMMUNITY OF PRACTICE (see pattern candidate **CO-01** in 5.2.2), the EMPOWERED COMMUNITY OF PRACTICE is obligatory for all selected development teams and comes with additional rights and regulations.

Example

The management fears that too many different expensive tools are used. They complain how various feature teams could be motivated to collaborate during the selection of tools.

Context

A usual COMMUNITY OF PRACTICE may not have enough power to justify far-reaching decisions. Regulatory standards suffer for consistent and powerful instruments to be enforced. Furthermore, a COMMUNITY OF PRACTICE with required but tedious topics may have not enough participants.

Problem

- **C-5:** *How to facilitate shared context and knowledge?*
- **C-26:** *How to align and communicate architectural decisions?*
- **C-27:** *How to manage and share knowledge about system components and their dependencies?*
- **C-38:** *How to facilitate standardization across agile teams?*
- **C-118:** *How to handle regulatory standards?*

Forces

- Usually a COMMUNITY OF PRACTICE works democratic. This is difficult if regulatory and cost intensive decisions are demanded.
- Responsibility of executives or workers with the relevant skills is required. Especially the role of the leader must be picked wisely in this extension.

Solution

The agile architecture decision-making model by Uludağ et al. [71] (see Figure 5.8) was used to decide whether a community is a COMMUNITY OF PRACTICE or an EMPOWERED COMMUNITY OF PRACTICE. Topics with a wide scope (number of development teams influenced by the topic) and high costs or regulations are domains for an EMPOWERED COMMUNITY OF PRACTICE, while all others are for a usual COMMUNITY OF PRACTICE. If needed, an EMPOWERED COMMUNITY OF PRACTICE is established by management.

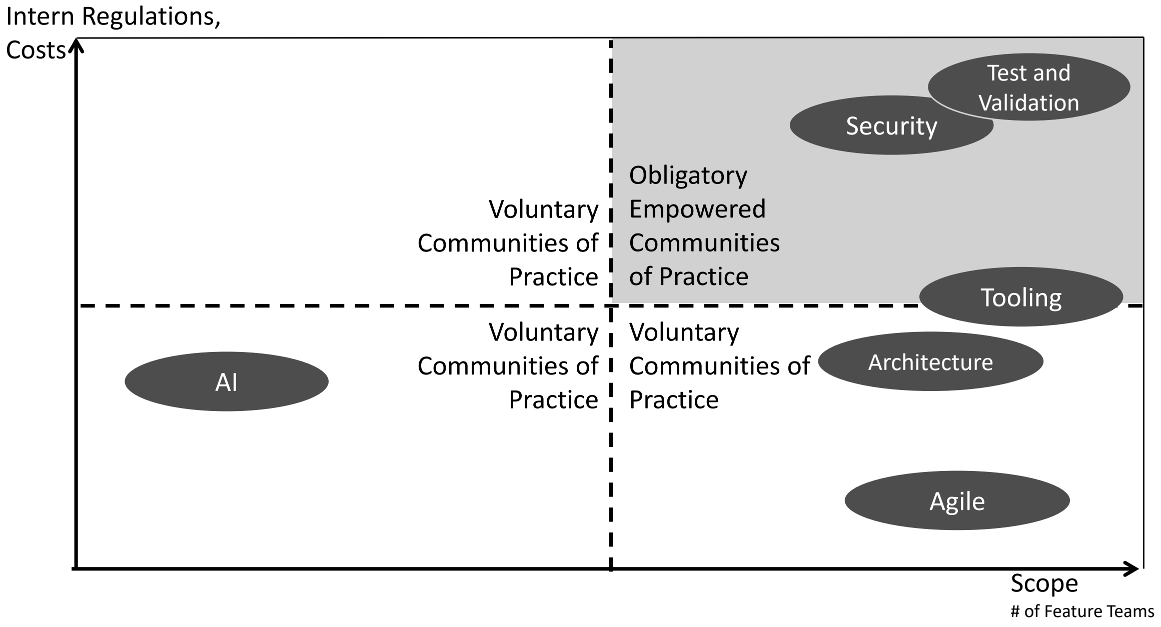
Intern Regulations, Costs
Test and Validation
Security
Voluntary Communities of Practice
Obligatory Empowered Communities of Practice
Tooling
Voluntary Communities of Practice
Voluntary Communities of Practice
Architecture
AI
Agile
Scope
# of Feature Teams

Figure 5.8.: Decision making model process with exemplary domains [71]

Usually, the EMPOWERED COMMUNITY OF PRACTICE works democratically. Some regulatory and cost intensive decisions require responsibility of executives. These should be made in the EMPOWERED COMMUNITY OF PRACTICE now. Consequently, a high responsibility follows. The leader is picked wisely by the management. He is an executive or skilled and experienced worker. The leader coordinates the EMPOWERED COMMUNITY OF PRACTICE and is empowered to take decisions valid for all development teams in teamwork with all participants.

Every selected development team sends at least one developer to the EMPOWERED COMMUNITY OF PRACTICE. The participation is obligatory. An EMPOWERED COMMUNITY OF PRACTICE can help to guarantee the compliance of the intern regulations and standards, as well as knowledge management in a feature-oriented structure.

Consequences

Benefits:

- Participation of all development teams in the decision making process is guaranteed.
- Guarantees networking through all selected feature teams.
- Guarantees a foundation for knowledge management for specific domains.
- Improves the quality for valid standards because experts are included in the decision making and standards are valid for the whole department.

Liabilities:

- Obligatory activities mostly have a lower motivation rate.
- Time intensive for the participants, because more participants are required and long discussions can appear.

See also

CO-01: COMMUNITY OF PRACTICE

5.2.4. Travelling

Pattern Overview	
ID	M-03
Name	TRAVELLING
Alias	Personnel Exchange
Summary	Developers change their team for one sprint to get deeper insights into technical and agile work of another development team.

Example

Two developers want to show each other their daily work.

Context

Single developers want to gain knowledge about the technical work of another team and wish to get insights in their agile work.

Problem

- **C-7:** *How to deal with incorrect practices of agile development?*
- **C-38:** *How to facilitate standardization across agile teams?*
- **C-59:** *How to establish a common understanding of agile thinking and practices?*
- **C-122:** *How to find time for training?*

Forces

- If the regular development team and the visited development team do not have the same sprint length and the sprint does not end at the same time, the traveler has an overload of work in between the times.
- Both teams may recognize a lack of competencies in their team during this sprint. On the other hand, especially for the planned tasks in the sprint, experts can travel to support the team.

Solution

Two developers can change positions for one or more sprints. The travelling is usually organized by the developer himself or the scrum master.

Developer A from team Alpha and developer B from team Beta normally do not work directly together. Their scrum masters propose them to change teams for one sprint length to become more familiar with the contents of the other one. They decide to take the chance and in one sprint they change teams to get insights in the work of the other one. That means for the duration of one sprint A develops in team Beta and B works in team Alpha. A is part of all meetings of the Beta team, while B is part of all meetings of the Alpha team during this time. After the sprint, they write down their experience and organize a meeting with all team members of the Alpha as well as of the Beta team. Also the product owners and the managers are invited. In the meeting they present their experience and give advice how the teams could change parts of their work and try to learn from one another. Especially the experienced differences between both teams regarding agile work, advantages and disadvantages they discovered, and their lessons learned are discussed.

Variants

- Developers can be sent to another team to support during busy times without having an exchange partner.
- Travelling can also be for longer, for example if developers are bored by their own work and want to get insights into another department.

Consequences

Benefits:

- Gain of knowledge about the technical work of other teams.
- Discover agile work of other development teams.
- Understanding how different work methods can be adapted.
- Possible support, if the partner is more experienced in the needed domain.

Liabilities:

- Loss of men-power during the sprint, if the partner is not as experienced in the needed domain.

See also

M-06: SHADOWING

5.2.5. Synchronized Calendar

Pattern Overview	
ID	V-04
Name	SYNCHRONIZED CALENDAR
Alias	One Calendar Approach, Synchronized Sprints
Summary	Several development teams share their frequent meeting times and locations to facilitate the coordination with other development teams.

Example

A COMMUNITY OF PRACTICE was installed and the two creators complain when they should conduct regular meetings. They do not know the frequent meeting times of the other feature teams.

Context

To facilitate SHADOWING (**M-06**), coordinate the participation in a COMMUNITY OF PRACTICE (**CO-01**) or a EMPOWERED COMMUNITY OF PRACTICE (**M-01**), share meeting rooms, allow TRAVELLING (**M-06**), and evolve transparency, a shared calendar can be used.

Problem

- **C-78:** *How to synchronize sprints in the large-scale agile development program?*

Forces

- The calendar should be always up to date, so every team is responsible to update the calendar.
- Responsibility for sharing in the different development teams, but teams may not prioritize the updating of the calendar so the coordination can be difficult.
- The room management may fail if not enough conference rooms are available and less time slots for meetings, especially longer ones can be offered.

Solution

All development teams share their meeting times (including locations) in an accessible calendar and update the calendar if data changes. Every color represents one team (see Figure 5.9). All teams work in sprints with a duration of two weeks, so meeting dates to sprint change times are shown by dividing the table in even and odd calendar weeks.

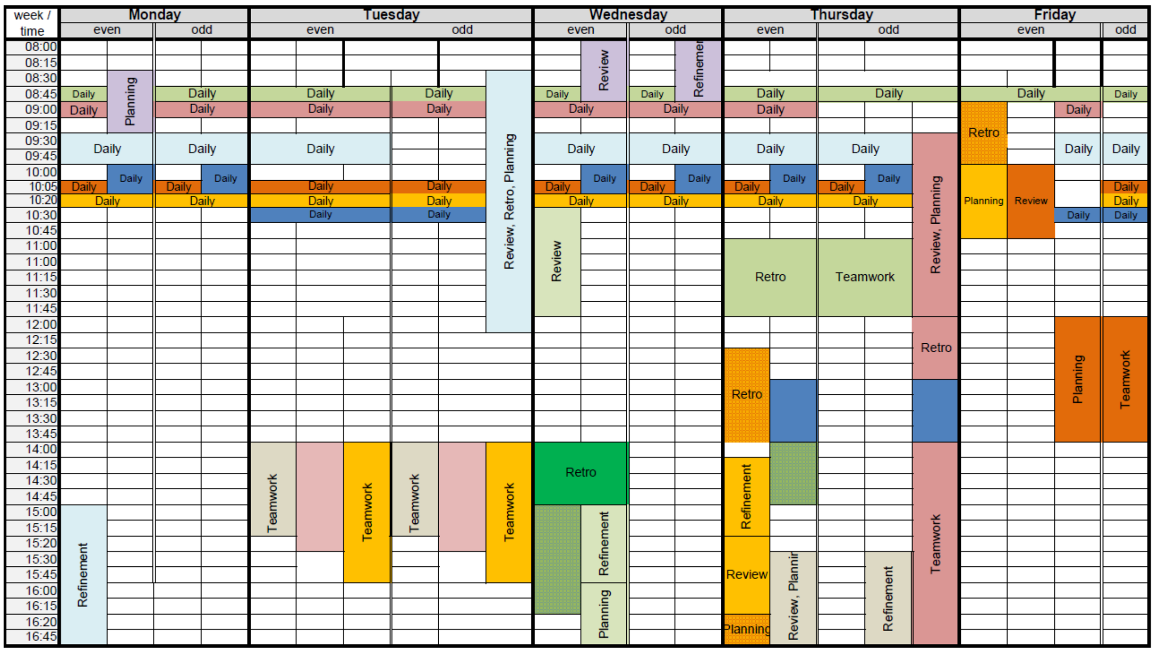

Figure 5.9.: Several teams share their meeting times in one calendar (daily scrum meeting, refinement meeting, planning meeting, review meeting, retrospective meeting)

Variants

- The synchronizing of sprints through a shared calendar adds more value to the communication during sprints. Several teams can synchronize their sprint length and sprint change times to increase their cooperation. The organization can be done central or local, depending on the preferences of the development teams and the management. Central organization facilitates the room management and the time management for cooperation (e.g. COMMUNITY OF PRACTICE, TRAVELLING, and more).
- Additional matching of sprint times (beginning, end, and duration) with suppliers.

Consequences

Benefits:

- Higher transparency because the meeting times are visible for everyone.
- Fast information transfer to stakeholders and interested members.

Liabilities:

- Responsibility for the care of the calendar is an additional work load.

5.2.6. Don't Combine Developers from Different Organizational Units in One Development Team

Pattern Overview	
ID	A-01
Name	DON'T COMBINE DEVELOPERS FROM DIFFERENT ORGANIZATIONAL UNITS IN ONE DEVELOPMENT TEAM
Alias	-
Summary	Developers from different organizational units sometimes work together in one development team. This was experienced as a negative practice and should be avoided.

Example

A new feature team is created with developers from different organizational units.

Context

During the transformation team structures have been reorganized, but the disciplinary structure was unchanged.

Problem

- **C-142:** *How to unite developers from different organizational units in one development team?*

Forces

- Changes in the organizational structure result in an administrative effort.

General Form

A new team is established. However, the organizational structure of the team is not reorganized. That means developers have different managers, work on different budget, have different work times, and more. Political discussions follow their daily life.

Consequences

Benefits:

- Fast personnel support if needed.

Liabilities:

- Political discussions rule the daily work life.
- Differences between the work units may lead to different work conditions (budget, room situation, personal preferences by the manager, and more).
- Developers do not feel united.

Revised Solution

When developers are reorganized in a new team structure, their disciplinary reorganization should follow too. That means, that all developers in the development team should have the same disciplinary manager and the same work conditions at their work place.

If the combination of developers in one organizational unit is not possible, it is recommended to facilitate their cooperation. A shared office as well as project rooms to spontaneously get together simplify their teamwork.

6. Discussion

This chapter outlines the key findings of this bachelor thesis. The most important scaling practices, found concerns, and good practices to address these are explained. Additionally, we present the effects of the adoption of agile development at the case organization. Later on, this chapter demonstrates the limitations of this work.

6.1. Key Findings

The following three scaling practices were identified as most important: First a dedicated work unit for agility, second COMMUNITY OF PRACTICE (**CO-01**) and their extension EMPOWERED COMMUNITY OF PRACTICE (**M-01**), and third a Product Owner Board. Furthermore, several concerns within large-scale agile transformation were investigated. We classified 54 as recurring. 12 of them were not investigated in previous research, but first through this study. Moreover, 17 success factors were documented as pattern candidates. One Principle, two Coordination Patterns, seven Methodology Patterns, six Viewpoint Patterns and one Anti-Pattern were found. Last, the impact of the new working at the case organization is discussed.

Scaling Practices

Three scaling practices have been identified as most important and analyzed:

Dedicated work unit for agility
A dedicated work unit for agile methodology supervised the agile work in the entire vehicle dynamics department. Guidance for all agile teams was provided by this work unit, for example training, workshops, and question times. In addition, this unit coordinates the scrum masters of all teams, technical as well as disciplinary, and guarantees close cooperation and shared focus. As a previous case study in another department showed, "comprehensive training courses and workshops ease the adoption since they provide a shared understanding of new practices, roles, and responsibilities" [68]. The referred special work unit enables this understanding and guarantees this also in future. Work units for agility were also identified before by Uludağ and Matthes [70] as Methodology Pattern Candidate "Agile Transition Team" [70]. The use of such a team was analyzed by them as being controversial and was also identified as Anti-Pattern [70]. We identified a dedicated work unit for agility as success factor during the transformation.

COMMUNITY OF PRACTICE and EMPOWERED COMMUNITY OF PRACTICE

A COMMUNITY OF PRACTICE is used to guarantee knowledge management and networking, while encouraging employees to open discussions. It combines employees from multiple development teams [13] in an informal network.
Additionally, an EMPOWERED COMMUNITY OF PRACTICE is an extension of the previously mentioned COMMUNITY OF PRACTICE. Instead of voluntary, this one is obligatory for selected development teams and comes with additional rights and regulations.
Paasivaara and Lassenius [47] identified characteristics of a successful COMMUNITY OF PRACTICE [47], which we previously compared to our adoption of them (see Table 4.4). In the case organization, several usages of an EMPOWERED COMMUNITY OF PRACTICE regulated the most expensive and far-reaching domains, as security or testing. The demand for this extension seems to be new. One reason could be that the higher management levels want to take the opportunity for a shared responsibility with the development teams. They experience this through the role of a coordinator who is enabled to make decisions in extreme situations and coordinates the activities. The coordinator in a COMMUNITY OF PRACTICE usually is selected by the participants [47], whereas in an EMPOWERED COMMUNITY OF PRACTICE, the higher management levels select the coordinator, and give him the power. The organization recommended that mainly executives can be coordinators in such practices, while a COMMUNITY OF PRACTICE shares the responsibility on all levels. Several uses of a COMMUNITY OF PRACTICE were recognized as a common scaling practice, which was also found by Kalenda et al. [35] and Paasivaara and Lassenius [47] in their empirical research. They rated them as a useful way to network, share knowledge, and motivate the employees by involving them in shaping their daily work life [35, 47]. A COMMUNITY OF PRACTICE offers high value and potential to achieve technical excellence [73]. The EMPOWERED COMMUNITY OF PRACTICE as extension of a COMMUNITY OF PRACTICE was first documented in recent research by Uludağ and Matthes [70] and is highly unpopular in literature.

Product Owner Board

The organization is scaling the role product owner in an unconventional way, even though this role often is used when scaling agile organizations in literature [14] and in several frameworks like LeSS (scaling the product owner role by scaling the product and ensuring an overall view [64]) or SAFe (here, a product owner owns a team backlog, while a product manager owns a program backlog [56]). The Product Owner Board is a circle including all product owners and higher management levels. They regulate work of product owners by using an additional backlog to prioritize tasks which are not related to one single feature but relevant for several features. Furthermore, such a board guarantees frequent networking for all product owners. Product Owner Boards are not listed as good practice by this work, since its success is not yet validated. Risk is that an additional backlog confuses work of development teams. Moreover, higher management levels can misunderstand their role since power given through the Product Owner Board is high. Their prioritization influences the work of all development teams - independently from the team-specific feature and their backlog. Due to the fact that higher management levels are not able to know the current details of the backlogs of single development teams at all times. The goal is to have a shared backlog in future and further potential of a Product Owner Board lies in the engineering of this. Currently, several development teams share one backlog when having a related feature. To have a proper overview they use tags (e.g. project names or vehicle parts) to organize it. In contrast to the Product Owner Board, a "Requirements management scaling" [35], identified by another case study by Kalenda et al. [35], organized and prioritized all tasks first in one shared backlog and pulled them later in single team backlogs for implementation of the tasks.

No common scaling practices regarding meetings for all development teams were used at the department. Kalenda et al. [35] for example identified "Scrum of Scrums", "Scaled Sprint demo/review", "Scaled planning", and "Scaled retrospective" [35] as scaling practices in literature and their research case. At the organization, some teams decided to share their sprint planning, sprint review, and sprint retrospective when having a combined feature, thus experiencing benefits from open discussions and information sharing. The demand for common meetings may arise when having a common backlog [48]. Nevertheless, meetings were recognized as an important part, especially in large-scale agile software development. Interviewees from piloting projects stated an increase in meetings when collaborating with other agile teams. As Moe et al. [43] mentioned, scheduled meetings are needed to coordinate in large-scale agile development and lead to more success of the overall agile program - particularly during adaption to agility. As mentioned before, the individual solution at the organization was that development teams decide on their own whether they have a need to collaborate with other development teams in meetings and therefore organize them together. Additionally, they use one common calendar together in which all meeting dates are shared to facilitate communication (see also pattern candidate **V-04:** SYNCHRONIZED CALENDAR).

Also, no strict implementation of a large-scale agile development framework, as the ones presented in Chapter 2.3.4, was conducted in the case organization. The department decided to use an individual solution with an own approach. This decision was mainly influenced by experiences by a partner department which strictly uses LeSS and found some limitations due to this. In the past decades, many studies have shown that adopting a more or less strict framework can lead to success (cf. step by step pattern language [25], adopting SAFe [46], adopting LeSS [68], and limits while adopting LeSS [48]). In contrast, Kalenda et al. [35], Fuchs and Hess [27], Rolland et al. [54] as well as this empirical study suggest to customize adoption and scaling of agile methods as far as possible in order to introduce a more appropriate large-scale agile development. This is consistent with focus on agile mindset in individual solutions [15].

Recurring concerns within large-scale agile transformation

Observations, interviews, and feedback talks highlighted concerns which the different roles at the organization faced in their everyday work life. We clustered all found concerns in several categories. Most concerns identified are regarding culture and mindset during agile transformation. The new focus on agile values and principles as a change in mindset were especially challenging in collaboration with traditional stakeholders and partner departments. Scrum masters more often mentioned wrong understanding of roles, right adaption of agile mindset, handling of false rumors, and personal doubts about employees - obviously concerns regarding the mindset and agile methodology. Also, Harders [30] identified many concerns for scrum masters are related to their mindset. Second leading is the category requirements engineering. Moreover, nine concerns in each of the following categories were identified: communication and coordination, software architecture, project management, and knowledge management. Management stated potential loss of knowledge and competencies as a particular fear - also before knowing whether loss of knowledge would come or not. Kuusinen et al. [37] found that knowledge management in scaled agile development is easier within teams, while more difficult outside a team. Additionally, the new style of working led to concerns for developers. For example, growing responsibility, need for more self organization, and fast growth of their own competencies was identified as a stress factor for some of them.
Our empirical research revealed 25 new concerns. Of these, 12 are classified as recurring:

C-116: *How to collaborate with the traditional world?*
C-117: *How to handle false rumors?*
C-118: *How to handle regulatory standards?*
C-119: *How to coordinate imperfect room situations?*
C-120: *How to guarantee the right understanding of roles?*
C-121: *How to reduce sideline activities?*
C-122: *How to find time for training?*
C-123: *How to simplify a complex tooling environment?*
C-124: *How to share the knowledge in the team to back-up failures?*
C-125: *How to distribute competencies across the team?*
C-126: *How to find the same wording for agile terms?*
C-127: *How to recognize the need for self-discipline?*

The correlation to existing literature has been explained in detail in Chapter 5.1. Unlike related research, this case study interviewed and observed different roles and concerns employees faced. The differentiation of various roles and their experiences was carried out first by our study.

Success factors within large-scale agile transformation

To address the observed concerns several good practices have been found and analyzed: Overall, candidates for two Principles, one Coordination Pattern, seven Methodology Patterns, six Viewpoint Patterns, and one Anti-Pattern were investigated.
We found that management demanded ways for knowledge sharing and ensuring competencies across all teams, which were established in several uses of the COMMUNITY OF PRACTICE (**CO-01**) and EMPOWERED COMMUNITY OF PRACTICE (**M-01**). Need for networking was also analyzed before [44]. Fuchs and Hess [27] detected the "Coordination of different organizational worlds" [27] as a barrier. Since a COMMUNITY OF PRACTICE was open for both, agile as well as non-agile participants, it facilitated the discussion of their concerns. COMMUNITY OF PRACTICE was identified as a good practice in other literature too [47,73]. Uludağ et al. [71] also analyzed the EMPOWERED COMMUNITY OF PRACTICE.

For inter-team coordination, SYNCHRONIZED CALENDAR (**V-04**) helped to organize collaborations. Additionally, TRAVELLING (**M-03**) and SHADOWING (**M-06**) secured the training of agile and technical developing between teams. Intra-team coordination was facilitated by BACKLOG GROOMINGS (**CO-02**) as well as BURNDOWN CHARTS (**V-06**), and the retrospective method STARFISH (**V-05**). As retrospective meetings are called the most important of all meetings [20], they had a high priority. Long-term as well as short-term technical and non-technical goals were coordinated by teams with a PROJECT PLAN (**V-03**), which are also used in traditional development, and a ROADMAP (**V-02**).

To guarantee the competency-advancement the principle CREATE T-SHAPED PEOPLE (**P-02**) was developed, and RADAR CHARTS (**V-01**) were established. Before starting the transformation through PILOTING (**M-02**) first experiences were gained in the department (cf. [18, 27, 32]). This showed the confrontation with traditional working teams and departments. To facilitate the involvement of non-agile parts of the company, as management or stakeholders, SHARE THE CHANGE (**M-04**) helped to inform everyone the same way about the transformation and changes in their daily work life. The communication of changes was also identified as a success factor by Dikert et al. [18]. They mentioned a higher transparency as an important reason. Stakeholders should COME TO OUR DEMOS (**M-05**), to get insights of the current work communicate via the team-mailbox (**M-07:** SHARE A MAILBOX). Consequently, transparency about work and team-structure increased(cf. [18]). Only one candidate for Anti-Pattern was identified: **A-01:** DON'T COMBINE DEVELOPERS FROM DIFFERENT ORGANIZATIONAL UNITS IN ONE DEVELOPMENT TEAM. This led to an unequal treatment of different members in a development team.

Additionally, developers, scrum masters, and product owners from teams where managers participated at training courses on agile development and mindset mentioned this as a success factor, because understanding of managers increased after coaching. Equally, Kalenda et al. [35] as well as Dikert et al. [18] identified support by management as a must.

Case organization

Overall, satisfaction of the employees increased when using agile work methods, 13 of 14 interviewees would prefer to continue agile development instead of returning to traditional work methods, even though new concerns occurred and developing agile was experienced as partly challenging. Although the development teams did not work completely self-organizing newly gained responsibility was also a motivator for developers and improved their worklife.

To improve agile work at this automotive organization, a rewarding system which is not individual but team-oriented could motivate development teams, as Dikert et al. [18] recognized a not team-oriented rewarding model as a concern. Implementation of such a system was not yet possible because workers council called for an individual evaluation for every single employee. Additional to the rewarding system present such a team-oriented rewarding system could be established. Also, the integration of personal advancement and agile goal of every single developer should be integrated. On the other hand, support of management must grow during the next months. Training and workshop sessions are already organized to improve agile mindset in management.

6.2. Limitations

Since this thesis used the design science approach, as explained in Chapter 1.3, limitations in both conducted fields result (literature research and qualitative study at one case organization). First limiting factor is the literature research limited by time. Thus, the main interest of this thesis was empirical research at the case organization and the literature research was conducted to get an overview about existing literature, existing state of research and similar cases and experiences. Since for this thesis a five-months case study was conducted, we were only able to observe the initial phase of the transformation and progression of the pilot project. Additionally, research for a longer period of actual working agile would offer a comparison between the current and the next states. Instead we had the chance to analyze the transformation itself with its concerns and success factors. So, the second limitation factor is the limited time frame for the case study. As the empirical research was only done at one specific and individual case organization restricted results were found and the results cannot be generalized automatically. To gain generalizability and more reliable results close comparisons to existing research were done. In order to prevent negative factors only dedicated to this company, observations at another agile developing department of this company were done. We compared them to literature and found no significant influences. The decision for qualitative instead of quantitative research limited the generalizability. Instead we were able to found rich in-depth details. As the goal of this thesis was to get deeper insights instead of a broad overview only a limited amount of interviews at the organization was conducted. Multiple different persons in different roles have been interviewed and feedback talks with several more stakeholders have been conducted. Even though, only the personal experiences of the participants are reported. Further, we were only able to document pattern candidates, but no pattern itself. This was because the study was conducted at only one company. Not only the validation, but also the evaluation of our pattern candidates is missing, because that was beyond the scope. We presented our results to the scientific as well as to the technical stakeholders and discussed the consequences of our work. Nevertheless, we did no organized evaluation, which would have been needed to fulfill the design science approach, because this was beyond scope of the present study.

7. Conclusion

This last chapter summarizes the main findings regarding the three research questions and gives an outlook on possible future work.

7.1. Summary

Since popularity of agile mindset, with its four values and 12 principles, is increasing, so is the demand for case studies in this domain. This is especially true in large-scale agile development, although this was not what agile methodologies were first designed for [22, 49]. This thesis aimed to investigate transformation to a cross-functional large-scale agile development, while analyzing occurring concerns, and to identify used good practices to address these. To summarize findings of this thesis, in the following, research questions mentioned in Chapter 1.2 will be answered here recapitulatory.

> **Research Question 1:** How does the large-scale agile transformation take place at the case organization?

The transformation from traditional software development methods to scaled agile development methods was investigated for five months. This was performed in a software development department in the field of vehicle dynamics development of an OEM (see Chapter 4). Without using a scaling framework the organization adapted agile development (see Chapter 4.2). Three main scaling practices were used: First, a special work unit, similar to an agile transition team, focusing on agility was established to coordinate agile development and work of the scrum masters. Second, COMMUNITY OF PRACTICE and EMPOWERED COMMUNITY OF PRACTICE were created to guarantee knowledge exchange and facilitate networking across development teams. Third, a Product Owner Board was installed. This circle, including all product owners and the higher management, regulated the work of product owners and managed an additional backlog for all development teams with especially important tasks. Additionally, more practices were registered to implement and coordinate agile transformation.

> **Research Question 2:** What are concerns within the large-scale agile transformation at the OEM?

The second research question aims to investigate what concerns occurred during the transformation explained above. With respect to the 79 challenges by Uludağ et al. [67], 25 more concerns were found during 14 interviews with different stakeholders. These are categorized and listed in Chapter 5.1 (see Figure 5.4 and 5.5). Overall, fifty-four recurring concerns were investigated, 12 of them were new discovered by us. Additional concerns were found, but characterized as not recurring. Four scrum masters, two product owners, three managers, and five development team members were interviewed. After analysis, we found that they faced different concerns throughout transformation. Also, the process of concerns was documented (see Figure 5.1). Summarizing, in the beginning mindset change and understanding of methodologies were the most important concerns, while later collaboration with non-agile dependencies as well as knowledge and competencies management gained relevance.

> **Research Question 3:** What are good practices to address the observed concerns within the large-scale agile transformation at the OEM?

To address observed concerns, 17 good practices have been identified and probed (see Chapter 5.2). Based on observation and interviews, these were documented as pattern candidates (see Chapter 2.1). We identified one Principle, two Coordination Patterns, one Anti-Pattern, seven Methodology Patterns, and six Viewpoint Patterns. These have been used by the development teams to shape agile development. A selection of most important pattern candidates was explained in Chapter 5.2, whereas additional pattern candidates can be found in the Appendix B. To sum up, success factors included training sessions, knowledge transfer, a conscious decision making process with teams, and slow scaling of agile development.

7.2. Future Work

As mentioned previously, this thesis was limited to a time frame of five months, so further work should investigate progress of transformation for a longer time period. The process of concerns over a longer scope of time could add value to current research. Also future researchers should be encouraged to compare the investigation of identified good practices in other organizations. As explained during the limitations of this thesis the evaluation of our pattern candidates as well as their validation to make them patterns remains. Both would add interesting value to current research. Furthermore, a stronger involvement of stakeholders which have dependencies with the case department could be interesting. During observation, some misunderstanding between agile and non-agile teams and a waste of work were realized because tasks were done twice. The complexity of change was felt on both sides, as stakeholders mentioned during feedback talks. A deeper view on stakeholders would add another insight on large-scale agile development in future work and highlight the understanding of their concerns. In addition to the suggested research in practice, moreover, a comparison between existing literature regarding scaling practices with and without a special framework would be interesting.

A. Appendix

The following chapter presents the questionnaires of the semi-structured interviews. First the general questions which all interviewees were requested to answer are presented, second the questionnaire which was special for the managers and third the questionnaire for the role of the scrum master, product owner and developers. Additional the resulting concerns are presented.

A.1. Interview Questionnaire

A.1.1. General Questions

All interviewees have been asked the general questions.

Background

1. What is your role in the agile software development?
2. How long have you been working in the field of agile software development?
3. How did you discovered agility?
4. How high is the level of scaling currently?
5. How many persons in your team are included in the agile development?
6. Who is not involved in the agile development?
7. Why is not everyone involved in the agile development?
8. What roles are included in your team?
9. How do you rate the cooperation within your team?
10. How do you rate the cooperation within the department?
11. What dependencies does your team has?

Transformation

1. What was the primary motivation for the transformation from you point of view?
   a) Improve Business/IT alignment
   b) Improve predictability of delivery
   c) Improve productivity
   d) Improve flexibility
   e) Manage changing priorities
   f) Reduce project risks
   g) Reduce project costs
   h) Improve software quality
   i) Improve software maintenance
   j) Improve team moral
   k) Reduce time-to-market
   l) Maximize value
   m) Improve coordination of distributed teams
   n) Improve project visibility

2. Who motivated you for the agile development?

3. How did they motivated you?

Challenges and Best Practices

1. What challenges did you face because of the agile development until nowadays?

2. How did you addressed the recurring challenges? What methodologies, practices or visualizations have been used?

Retrospective

1. What advantages arose since the start of the agile development?

2. What disadvantages arose since the start of the agile development?

3. Would you prefer to continue agile development or go back to traditional work methods?

A.1.2. Questionnaire for Manager

The managers were requested to answer some more role-specific questions.

Transformation

1. Who motivated your team for the agile transformation?
2. How have your team been motivated?
3. How is the agile transformation rolled out?
4. How do you scale the agile development?

Challenges and Best Practices

1. What challenges did you face because of the agile development until nowadays?
2. What role-specific concerns did you monitor?
3. How did you addressed the recurring challenges? What methodologies, practices or visualizations have been used?
4. What good practices have you recognized?
5. What challenges could not be solved until nowadays?

Success Stories

1. What are current success stories?
2. What are important factors for the success of the actual program?
3. What advise do you have for other agile programs?
4. What should be avoid during agile transformations?

A.1.3. Questionnaire for Product Owner, Scrum Master and Developer

Concerns and Best Practices

During the second part of the interview session a selection of 56 of the 79 challenges found by Uludağ et al. [67] have been presented to the interview partner. If a challenge was annotated as occurring by the interviewee, they had the chance to mention when and how often the concern was recognized. This is the selection:

- **C-1:** *How to coordinate multiple agile teams that work on the same product?*
- **C-2:** *How to consider integration issues and dependencies with other subsystems and teams?*
- **C-4:** *How to deal with doubts in people about changes?*
- **C-5:** *How to facilitate shared context and knowledge?*
- **C-6:** *How to manage technical debts?*
- **C-7:** *How to deal with incorrect practices of agile development?*
- **C-8:** *How to ensure that non-functional requirements are considered by the development team?*
- **C-9:** *How to find the right balance between architectural improvements and business value?*
- **C-10:** *How to create precise requirement specifications for the development team?*
- **C-13:** *How to share a common vision?*
- **C-14:** *How to create a proper upfront architecture design of the system?*
- **C-15:** *How to elicit and refine requirements of end users?*
- **C-16:** *How to deal with increasing workload of key stakeholders?*
- **C-17:** *How to establish self-organization?*
- **C-18:** *How to split large and complex requirements into smaller requirements?*
- **C-19:** *How to deal with internal silos?*
- **C-20:** *How to facilitate communication between agile teams and other teams using traditional practices?*
- **C-21:** *How to manage dependencies to other existing environments?*
- **C-22:** *How to balance short-term and long-term goals?*
- **C-23:** *How to establish a common scope for different stakeholder groups?*

- **C-24:** *How to create team spirit and trust among agile teams?*
- **C-25:** *How to manage and integrate heterogeneous subsystems of different development teams?*
- **C-26:** *How to align and communicate architectural decisions?*
- **C-27:** *How to manage and share knowledge about system components and their dependencies with stakeholders?*
- **C-28:** *How to communicate business requirements to development teams?*
- **C-31:** *How to deal with geographical distance between agile teams?*
- **C-33:** *How to build trust of stakeholders in agile practices?*
- **C-35:** *How to define clear and visible priorities?*
- **C-37:** *How to create lightweight documentation?*
- **C-38:** *How to facilitate standardization across agile teams?*
- **C-39:** *How to establish a culture of continuous improvement?*
- **C-41:** *How to deal with unplanned requirements and risks?*
- **C-43:** *How to enforce customer involvement?*
- **C-44:** *How to deal with communication gaps with stakeholders?*
- **C-45:** *How to deal with black and white mindsets?*
- **C-46:** *How to deal with closed mindedness?*
- **C-47:** *How to deal with higher-level management interference?*
- **C-49:** *How to deal with increased efforts by establishing inter-team communication?*
- **C-50:** *How to deal with lacking sense of ownership responsibilities for developed services?*
- **C-55:** *How to create a teamwork centric rewarding model?*
- **C-56:** *How to define clear roles and responsibilities?*
- **C-58:** *How to deal with loss of management control?*
- **C-59:** *How to establish a common understanding of agile thinking and practices?*
- **C-60:** *How to create an estimate user stories?*

- **C-63:** *How to explain requirements to stakeholders?*
- **C-65:** *How to deal with office politics?*
- **C-66:** *How to foster technical excellence?*
- **C-67:** *How to encourage development teams to talk about tasks and impediments?*
- **C-69:** *How to establish requirements verification?*
- **C-70:** *How to define high-level requirements a.k.a. epics?*
- **C-71:** *How to measure the success of the large-scale agile development program?*
- **C-72:** *How to consider required competencies when assigning teams to tasks?*
- **C-73:** *How to deal with decreased predictability?*
- **C-74:** *How to empower agile teams to make decisions?*
- **C-77:** *How to build an effective coaching model?*
- **C-78:** *How to synchronize sprints in the large-scale agile development program?*

A.2. Results

The results of the survey about the concerns have been analyzed and sorted by frequency in Figure A.1 and Figure A.2. Labeled in green are concerns mentioned proactively (see Questionnaire A.1.1), while labeled in orange are concerns which were selected from the concerns presented to the interviewee (see Questionnaire A.1.3).

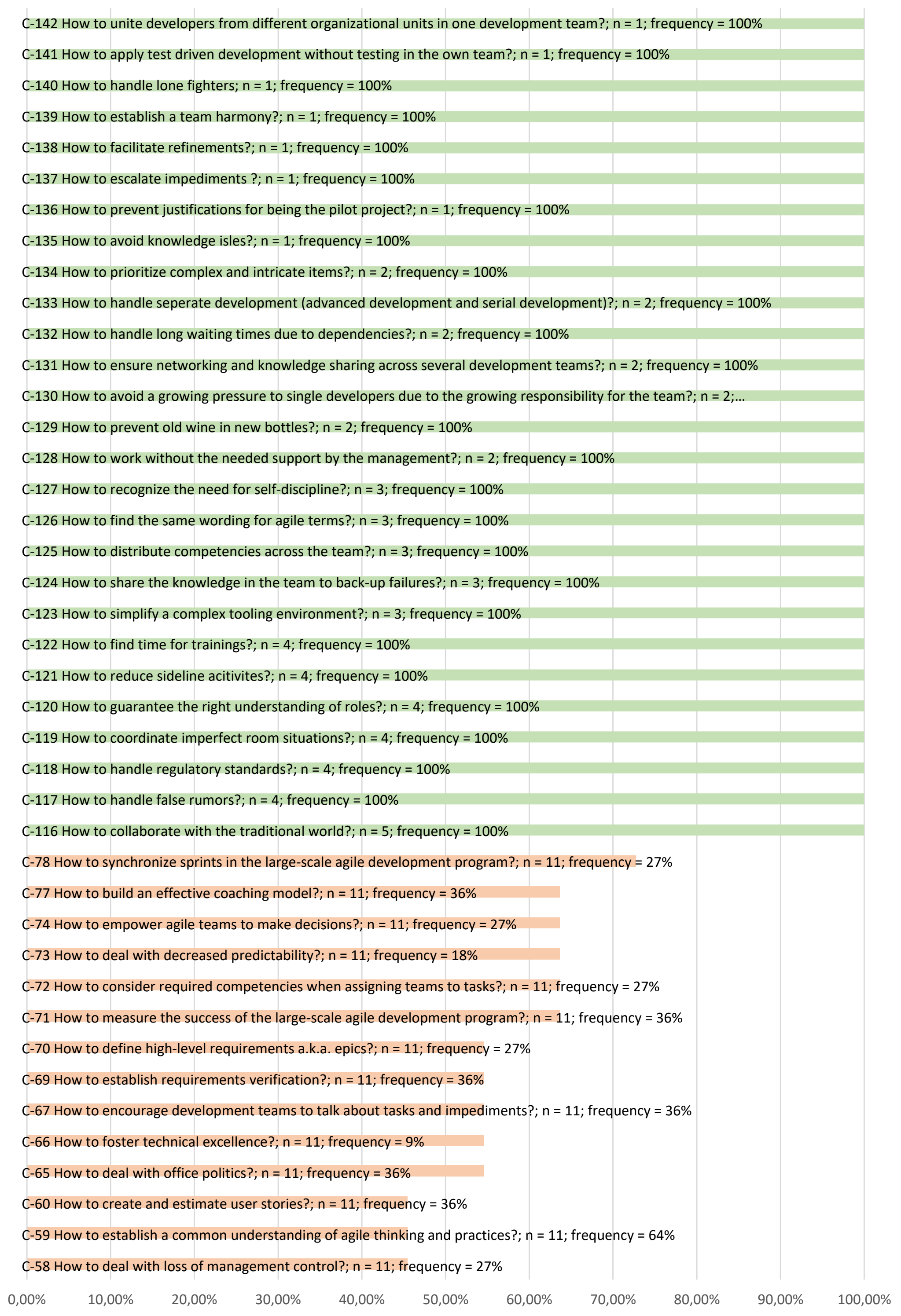

Figure A.1.: Occurring concerns sorted by frequency (Part 1)

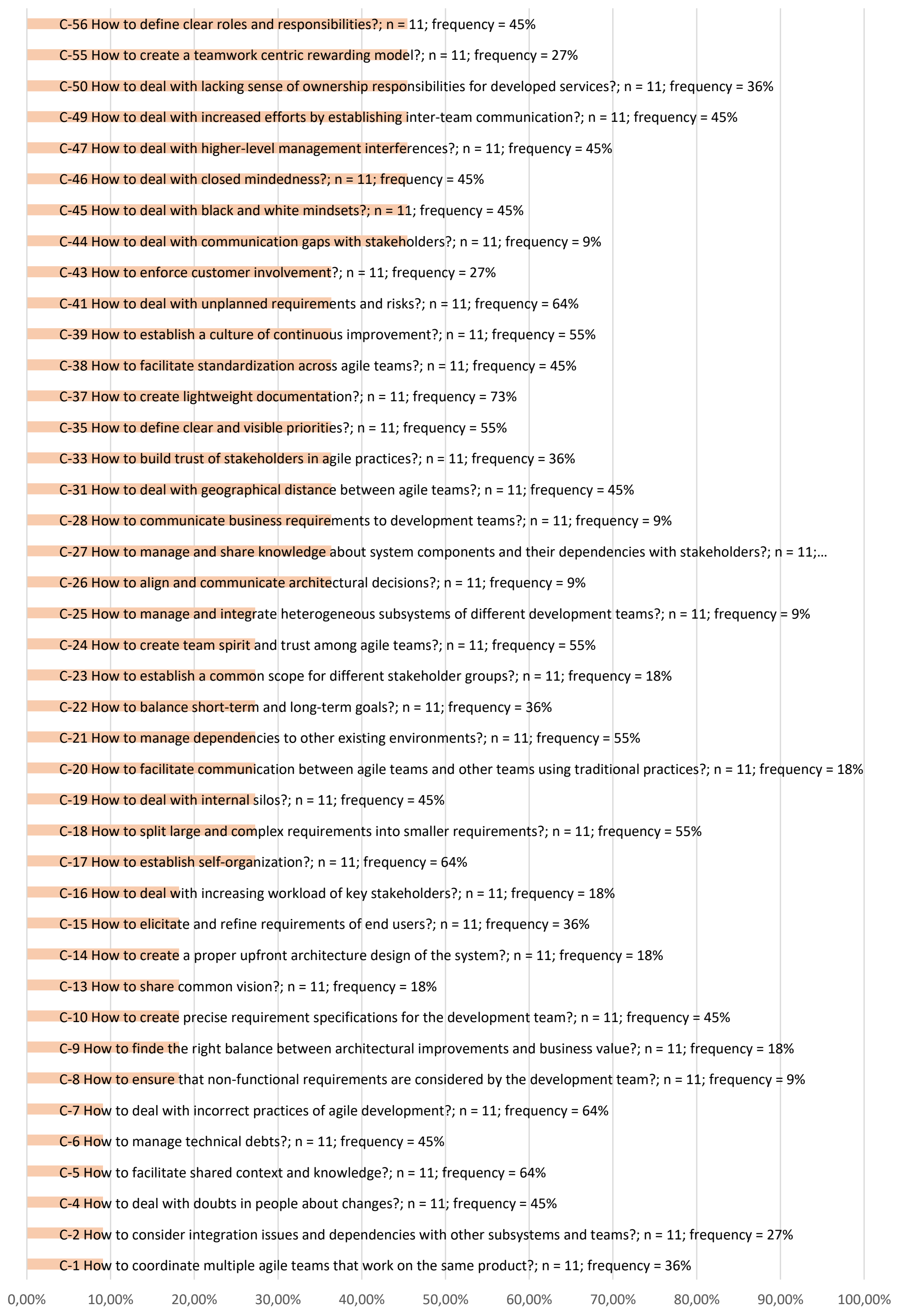

Figure A.2.: Occurring concerns sorted by frequency (Part 2)

B. Appendix

B.1. Documentation of Identified Pattern Candidates

B.1.1. Backlog Grooming

Pattern Overview	
ID	CO-02
Name	BACKLOG GROOMING
Alias	Backlog Refinement
Summary	Product BACKLOG GROOMING is the continuous grooming process of the product backlog. Product owner, development team, and scrum master collaborate to optimize the clear representation of the backlog.

Example

The product owner needs help with the organization of new tasks in the backlog, while the developer do not understand the way the product owner organized their backlog before.

Context

A confusing backlog with too many tasks is neither a good base for planning meetings nor useful for prioritization of new tasks. Furthermore, the product owner benefits from the help of the developers and the developers increase their understanding for the product owners work.

Problem

- **C-35:** *How to define clear and visible priorities?*
- **C-67:** *How to encourage development teams to talk about tasks and impediments?*

Forces

- BACKLOG GROOMING showed best effect at the case organization when conducted regular and with many participants. Result is one more time intensive meeting in the tight schedule of all employees.
- People want so reorganize the entire backlog and split all tasks at once.

Solution

The BACKLOG GROOMING was used to organize the backlog [5]:

1. Create tasks and refine stories
2. Split large tasks into smaller ones
3. Prioritize tasks

During the observation different variants have been identified. The most common was that the product owner already prepared step one and the development team was requested to help him with step two. In the third step the product owner suggested his most important next tasks grounded on long- and short-term goals (see also **V-03:** PROJECT PLAN) and discussed these with the development team. Some teams also estimated the tasks and were able to reduce this step in the planning meeting.

Variants

Backlog Refinement, Backlog Estimation

Consequences

Benefits:

- Better overview about the backlog for all team members.
- Facilitates sprint planning meetings, because the backlog is understandable as well as known to all team members.
- Facilitates the organization of the backlog for the product owner, because the developers assisted him.

Liabilities:

- The time intensive meeting is experienced as an additional work load by some developers.

B.1.2. Piloting

Pattern Overview	
ID	M-02
Name	PILOTING
Alias	-
Summary	To ascertain the environment for agile development, gain first experience and experiment a new idea a pilot project is started.

Example

The department recognizes the new popularity of agile development and adopts agile practices only in a few teams, because they are not sure about the practicability for their department.

Context

To test how agile development can be adopted at the case organization sample teams start to work agile.

Problem

- **C-81:** *How to enable change from process to product orientation?* (found by Uludağ and Matthes [70])

Forces

- If the experiment fails, new pilots need to start.
- Pilots often need to justify in their department, for working different and creating additional work to others without knowing whether it will improve their development.

Solution

Organizations start a pilot, before they decide, whether agile development may be the right solution for their needs. First individual and later scaled, several development teams start to work agile and recognize mistakes to avoid these in the future. Additionally, success factors are documented to use them as suggestions for future development.

Before a pilot finishes, the outcome is not defined. That means a failing is possible and requires a new pilot afterwards. A succeeding pilot is the positive approval for the agile working to the management. However, the differentiation in succeeding and failing is often difficult, because many factors influence the outcome. Consequently, all factors about the performance of the pilot must be visible to the management.

Consequences

Benefits:

- Early experiences by pilots can be used for the transformation and help to avoid the recurring of concerns.
- PILOTING can be used as a training for inexperienced people by using SHADOWING (**M-06**).
- Fail fast, fail cheap (cf. [32]) is a motivation to fail as long as it does not influences other projects. It motivates to fail in initial phase, which PILOTING is.
- Successful experiments motivate developer to continuously improve their work.

Liabilities:

- Time intensive, because the installation takes time.
- Failing experiments can demotivate developer.
- During PILOTING, developers often experienced the need to justify for their novel way of working (see also **C-136:** *How to prevent justifications for being the pilot project?*).

B.1.3. Share the Change

Pattern Overview	
ID	M-04
Name	SHARE THE CHANGE
Alias	Change Drive
Summary	Involve all employees and stakeholders, and communicate the process, goal and motivation of the change during a workshop.

Example

One departments decides to adopt agile development. The management is not sure how to communicate the changeover. A workshop series solves the problem.

Context

During a transformation the work routine for employees and stakeholders is influenced by the change. To include all employees in change, a well-organized workshop with time for discussion helps the employees to get insights in the transformation.

Problem

- **C-4:** *How to deal with doubts in people about changes?*
- **C-59:** *How to establish a common understanding of agile thinking and practices?*
- **C-116:** *How to collaborate with the traditional world?*
- **C-117:** *How to handle false rumors?*
- **C-128:** *How to work without the needed support by the management?*

Forces

- It takes time for stakeholders to understand the change and to find their new place in the organization (e.g. new contact persons).
- It is cost and time intensive to inform and include all employees during the change in a large organization.
- The performance of all workshops takes time and needs personal capacities.

Solution

Workshop series organized by a team of employees with the direct assistance of the (higher) management. The workshop should collect all stakeholders and motivate them to be part of the transformation. Even stakeholders which are not directly involved in the change should be informed at the workshop to get a higher transparency.

Possible content for the workshop:

- Show motivation for the change (why do we need a transformation?);
- who organized the transformation;
- what will change;
- what already changed;
- what will not change;
- vision;
- explain new work methods simply to the participants (to all participants, also the ones which will not work with the new work methods);
- deviate a common roadmap.

Consequences

Benefits:

- Everyone is involved in the changeover.
- Higher motivation because all employees have a voice.
- Transparency due to the explanation of the change to everyone in the same way.
- Promote dialog between organizers of the transformation and stakeholders.

Liabilities:

- Cost as well as time intensive during the organization and implementation.

B.1.4. Come to Our Demos

Pattern Overview	
ID	M-05
Name	Come to our Demos
Alias	Visit Us
Summary	All stakeholders and other interested colleagues are invited to join the meetings of a development team. They share the used methodology and their technical work.

Example

Stakeholders ask the team to present their outcome and invite them to present their work done. Instead of visiting the different stakeholders at once, the team decides to invite them.

Inexperienced scrum masters ask more experienced scrum masters how their meetings are conducted.

Context

During a transformation stakeholders lack the knowledge of the agile mindset and distrust the agile practices, while requesting the presenting of results.

Moreover, new agile teams (developers, scrum masters, and product owners) lack agile experience in the first sprints.

Problem

- **C-7:** *How to deal with incorrect practices of agile development?*
- **C-33:** *How to build trust of stakeholders in agile practices?*
- **C-59:** *How to establish a common understanding of agile thinking and practices?*

Forces

- Stakeholder as well as some new developer using agile practices may distrust new development methods and do not have time to come to all meetings.
- Visitors may not understand all content which is discussed in the meetings.

Solution

Interested Colleagues (for example inexperienced new developer, scrum master and product owner) can join the meetings (especially the daily scrum meeting, the planning and the review meeting) of experienced development teams to get insight not only into the technical but also the agile work (see also **M-06:** SHADOWING).

Managers and stakeholders should use the review meeting to get an overview of the current work of a development team instead of request updates. To be as transparent as possible a agenda can be sent to the visitors. Consequently, they can check, whether the content of the review meeting is relevant for them. To get the highest possible outcome of the meetings, demos should be well structured and easy understandable for all visitors. The development team should think about the content which is presented previously.

Variants

- COME TO OUR DEMOS to notice the used agile methodologies.
- COME TO OUR DEMOS to get insights in our technical work.

Consequences

Benefits:

- Teams share their knowledge automatically during the demo with their stakeholders.
- Gives visitors the chance to get insights in the agile mindset. Further, they can ask questions regarding the agile mindset direct to the agile teams.
- Saves time, which would have been used for the presentation of request updates.
- Transparency regarding all invited stakeholders.

Liabilities:

- Time intensive for visitors.

See also

M-06: SHADOWING

B.1.5. Shadowing

Pattern Overview	
ID	M-06
Name	SHADOWING
Alias	Following, See and Go
Summary	To get insights in the work of others, one colleague follows another one to meetings like a shadow and observes the colleagues work.

Example

A new colleague wants to get insights in the work of an experienced scrum master.

Context

Stakeholder, managers and others want to get insights in the work of agile teams.

Problem

- **C-5:** *How to facilitate shared context and knowledge?*
- **C-47:** *How to deal with higher-level management interference?*

Forces

- The sharing of knowledge is a time intensive task, because two parties are included: the teacher and the learner. Resulting, time capacities of both parties are consumed during the usual way of teaching. One solution is to have just one capacity consumed by showing the learner what the teacher does during his job.

Solution

Interested colleagues can follow special related roles in the agile work to understand their way of working. The person following acts like a shadow and observes meetings, without interrupting them. Teams should not feel disturbed, so it is important that a shadow does not interact with the teams during a meeting. The person which is followed can explain open topics to his shadow, if needed after meetings.

Experienced colleagues (especially scrum masters and product owners) use shadowing to help inexperienced scrum masters and product owners to improve their work in the initial phase of the transformation. Moreover, managers follow a scrum master or product owner. Additionally, inexperienced developer can follow the members of a development team, or stakeholder follow their partner.

Consequences

Benefits:

- Insights in the work without disturbing a team.

Liabilities:

- Time intensive for the shadow.
- Teams may act not as usual when the shadow participates.

See also

M-05: COME TO OUR DEMOS

B.1.6. Share a Mailbox

Pattern Overview	
ID	M-07
Name	SHARE A MAILBOX
Alias	-
Summary	A development team installs an own mailbox for impulsive assignments and requests, to facilitate the prioritization and establish transparency for all team members and the product owner.

Example

The developers complain that they get many requests from different stakeholders during one sprint. They do not know how to prioritize the requests and feel lost.

Context

Development teams working for projects in the serial production, often get spontaneous, urgent, and important requests or extra specification during a sprint. Many times, these are addressed to single developers. Developers may not know how to prioritize the request.

Problem

- **C-41:** *How to deal with unplanned requirements and risks?*
- **C-47:** *How to deal with higher-level management interference?*
- **C-49:** *How to deal with increased efforts by establishing inter-team communication?*

Forces

- Importance of different assignments cannot be prioritized by single developers and stress the sprint. Consequently, technical debts follow a sprint.
- Planned tasks cannot be solved because of secondary occupations.

Solution

The development team and their product owner install a mailbox for all spontaneous requests and assignments. Stakeholders are instructed to use this shared mailbox instead of the personal mailbox of the single developers for urgent requests. The product owner takes care of the mailbox and prioritizes new tasks into the product backlog. Potential changes are shared in the daily scrum meeting. If the product owner cannot prioritize alone, he asks for help in the development team and reorganizes the tasks in the backlog.

Consequences

Benefits:

- Higher transparency for the product owner because he notices all requests. Consequently, all tasks are ordered in the backlog by the product owner.
- Less stress for single developers.

Liabilities:

- Work boost and potential bottleneck for a product owner, because of the effort to prioritize the mailbox.
- Personal contact to stakeholders may slowdown for developers.

B.1.7. Radar Chart

Pattern Overview	
ID	V-01
Name	RADAR CHART
Alias	Spider Chart
Summary	The RADAR CHART represents the competencies in a development team. It compares the actual and target status of competencies.

Example

The development team does not agrees about competencies in their own team.

Context

A development team wants to compare the knowledge management and the increase of competencies in a period or evaluate the change of development team members and uses a RADAR CHART to rate needed competencies and available competencies.

Problem

- **C-5:** *How to facilitate shared context and knowledge?*
- **C-39:** *How to establish a culture of continuous improvement?*
- **C-72:** *How to consider required competencies when assigning teams to tasks?*

Forces

- Self-assessment may vary from developer to developer.
- The chart needs to be up to date every time to be useful.

Solution

The development team comes together to analyze which knowledge categories are needed, to fulfill their feature in the best way. Next, they rate their own competencies in the different needed knowledge categories. Everyone rates himself and all ratings are merges in a team RADAR CHART (actual state). Furthermore, a competency radar chart for the product (target state to solve the tasks regarding the product) is generated to compare both RADAR CHARTs (see Figure B.1).

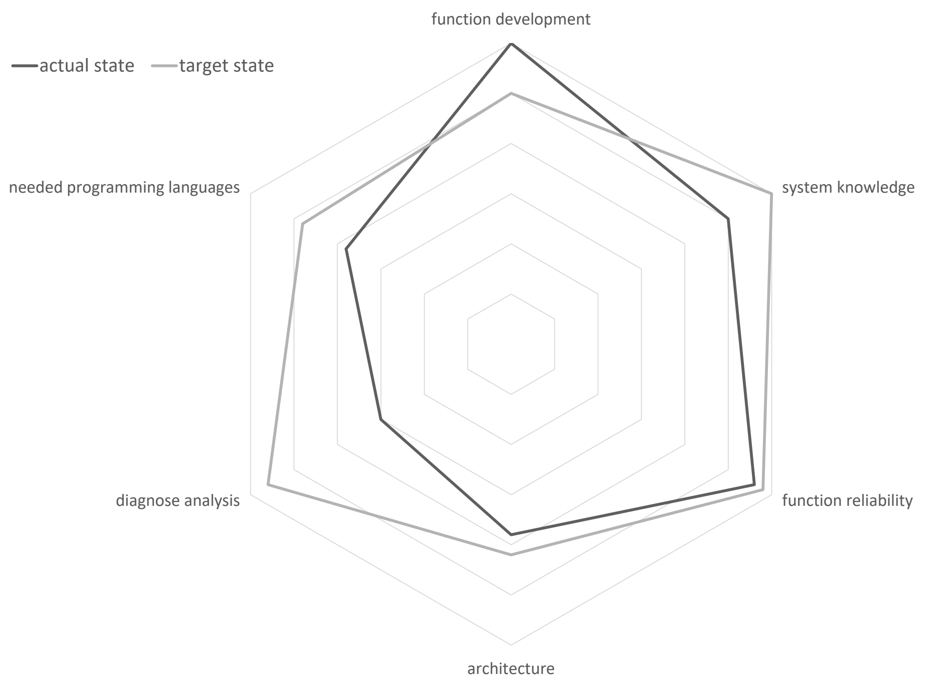

Figure B.1.: Example radar chart for the division of competences

Consequences

Benefits:

- Competencies are clarified for the product owner.
- Simplifies knowledge building for the team.
- Helps to create a ROADMAP (**V-02**) and optimize the distributed competencies through an increase of knowledge management and control the progress.
- Improves the self-image of every developer.

Liabilities:

- Time intensive, because each developer must categorize his competencies.

See also

V-02: ROADMAP

B.1.8. Roadmap

Pattern Overview	
ID	V-02
Name	ROADMAP
Alias	-
Summary	A ROADMAP for the team to focus on the next needed tasks for the product and the continuous improvement of the team itself.

Example

The development team argues how to manage their individual goals together with their technical team-oriented goals.

Context

Development teams organize their goals in a ROADMAP to have a manual including long-term and short-term goals.

Problem

- **C-9:** *How to find the right balance between architectural improvements and business value?*
- **C-13:** *How to share common vision?*
- **C-22:** *How to balance short-term and long-term goals?*
- **C-39:** *How to establish a culture of continuous improvement?*

Forces

- The managing of short-term and long-term goals is often not easy understandable for developers.
- The coordination of technical as well as personal goals for all team members is challenging.
- Developers do not know how to communicate their individual goals.

Solution

Development team, product owner and scrum master create a ROADMAP together, including important short- and long-term goals to guarantee the fulfillment of these regarding the development team and the technical product. Different layers can represent different aspects for the product, roles or stakeholder. The time line is the foundation of the ROADMAP (see Figure B.2).

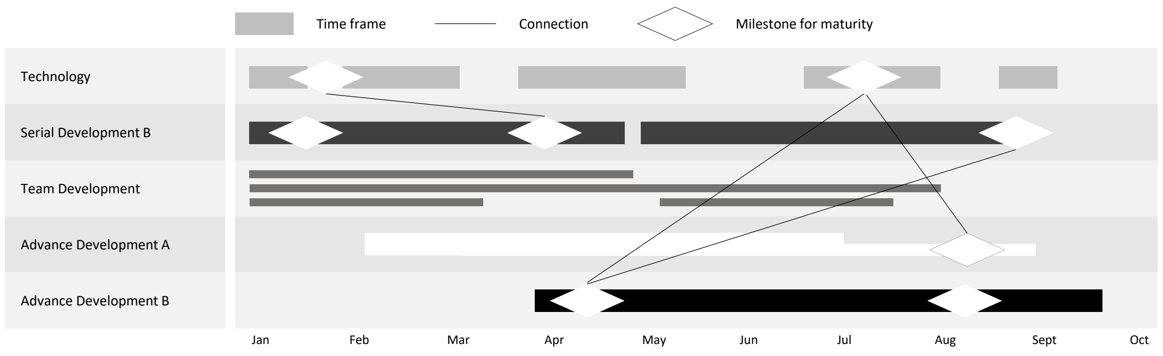

Figure B.2.: Example ROADMAP

Variants

As Phaal and Muller [50] explained in their architectural framework for roadmapping, several types of roadmaps exist. Roadmaps can consist of more or less influence factors, including roles as well as milestones.

Consequences

Benefits:

- Overview about long-term and short-term goals for the team.
- Overview including individual, technical, agile and more different types of milestones.
- Simplifies reporting to the management: show ROADMAP and check growth.

Liabilities: *None known.*

See also

V-03: PROJECT PLAN

B.1.9. Project Plan

Pattern Overview	
ID	V-03
Name	PROJECT PLAN
Alias	Calendar Overview
Summary	During a planning meeting, the product owner presents a project plan to justify his prioritization of the sprint backlog and gives the development teams an overview about relevant tasks beyond the current sprint.

Example

The developers request their product owner to share not only the direct upcoming tasks, but also the long-term tasks.

Context

The organization of technical milestones and processes, done by the product owner, should be visible and understandable for the development team.

Problem

- **C-22:** *How to balance short-term and long-term goals?*
- **C-35:** *How to define clear and visible priorities?*

Forces

- The product owner must summarize all dates in one table, in a structured manner, that can lead to a loss of information.

Solution

The product owner brings a PROJECT PLAN as overview to a planning meeting, to inform the development team about the deadlines for the next weeks and new appointments. He marks the relevant dates for the next few sprints and explains why he prioritized which task in the current sprint. The development team members can request, if something is not clear and have access to the PROJECT PLAN the entire time. The calendar is organized as a table with columns as days/weeks/months and rows as current projects/series (see Figure B.3). For a better understanding different types of deadlines can be colored (example clustering for these milestones: model, implementation, test, documentation and more).

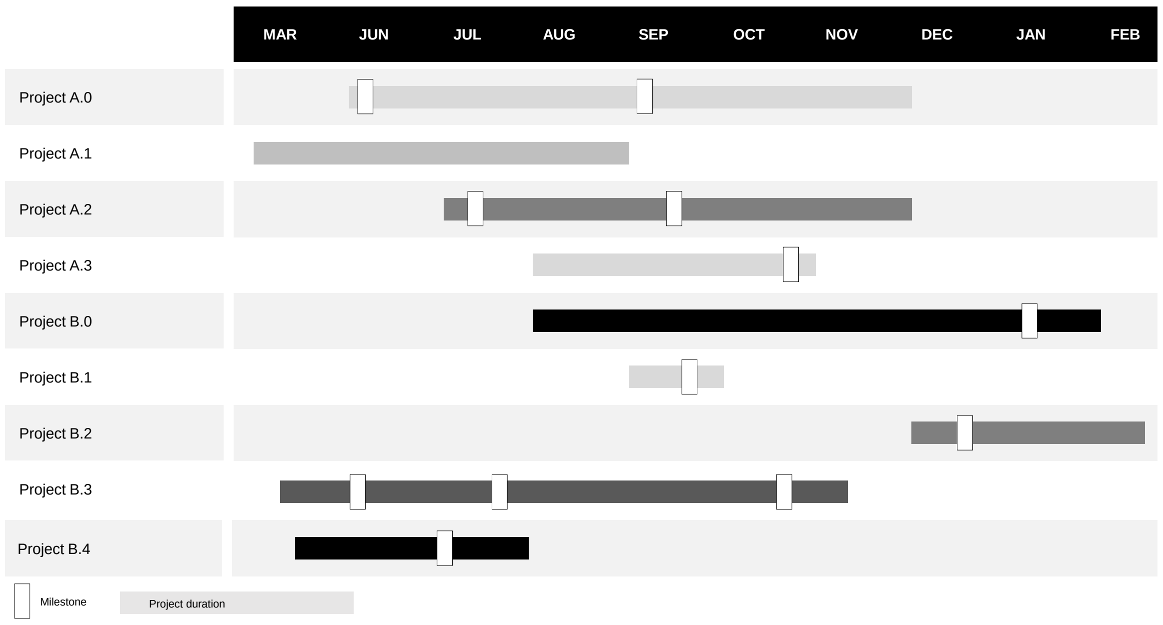

Figure B.3.: Exemplary project plan with milestones (e.g. technical maturity)

Variants

The calendar can include all useful information for the team. However, it is important to limit the information to guarantee a clear overview. Calendar overviews can vary for different teams.

Consequences

Benefits:

- Clear overview for the teams about the technical goals.
- Strengthens understanding of sprint backlog for the team.

Liabilities:

- Temporal expenditure of the plan leads to more work load for the product owner.

See also

V-02: ROADMAP

B.1.10. Starfish

Pattern Overview	
ID	V-05
Name	STARFISH
Alias	-
Summary	STARFISH is a retrospective technique to help team members to share their thoughts in a retrospective meeting to improve their teamwork.

Example

The scrum master recognizes, that some developers feel uncomfortable expressing themselves during retrospective meeting.

Context

Usually in a retrospective meeting, the team gets asked what went well, not so well, and what could be improved. To provide more facets the starfish retrospective can be used.

Problem

- **C-59:** *How to establish a common understanding of agile thinking and practices?*
- **C-137:** *How to escalate impediments?*

Forces

- Team member sometimes feel not well sharing their (negative) thoughts about the work within the team, while feeling uncomfortable addressing them.

Solution

A sea-star with five keywords is shown to the team members: Stop, Less, Keep, More, and Start (see Figure B.4). Every team member writes down their thoughts regarding the five categories at the starfish board and afterwards every point is discussed.

Stop doing: These activities do not bring a value to the team and should be avoided.
Less of: Something that works, but the value is not high enough. So the effort is higher than the output.
Keep doing: Something that works well and the value is recognized. Usually, good practices the team members want to keep.
More of: Something which is already done, but could bring more value if done more often. So the team should focus more on it.
Start doing: New ideas that could fit into the team work.

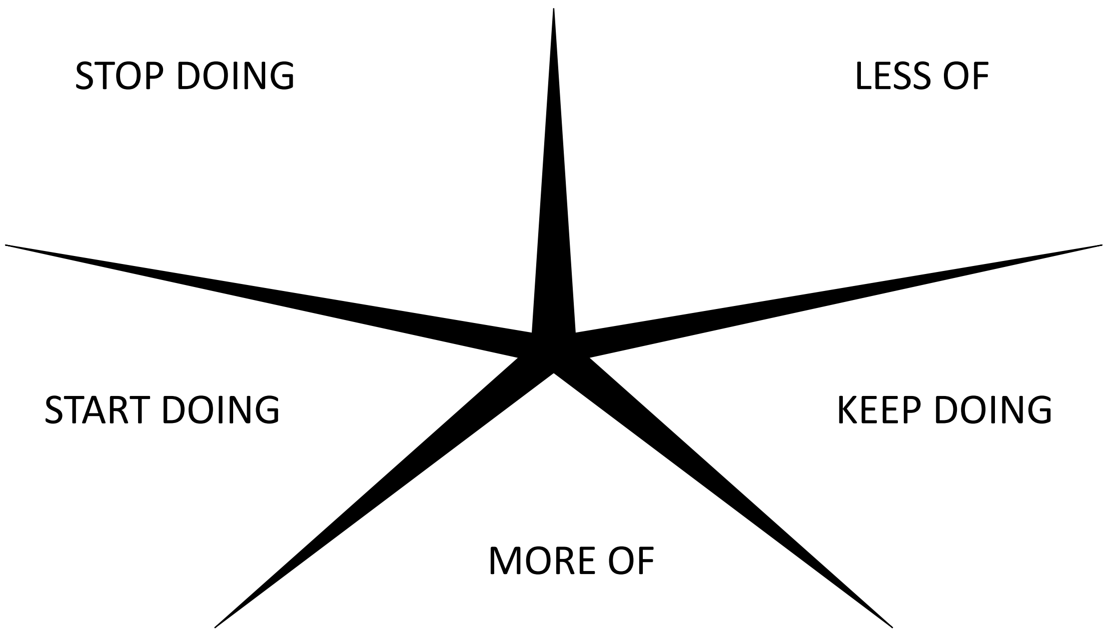

Figure B.4.: Empty STARFISH structure

Consequences

Benefits:

- More options for teams during a retrospective meeting.
- Facilitate discussions about concerns for developers.

Liabilities:

- Some teams or single developers may not feel an additional value.

B.1.11. Burndown Chart

Pattern Overview	
ID	V-06
Name	BURNDOWN CHART
Alias	-
Summary	The BURNDOWN CHART gives an overview how much time is left in a sprint and how much work is already done.

Example

A team fails frequently in solving their sprint tasks. The developers discuss possible reasons.

Context

Some development teams do not solve all their sprint goals. To understand how much work was done during a sprint in what time, the BURNDOWN CHART compares the real tasks done with the ideal amount of tasks done.

Problem

- **C-41:** *How to deal with unplanned requirements and risks?*
- **C-121:** *How to reduce sideline activities?*
- **C-130:** *How to avoid a growing pressure to single developers due to the growing responsibility for the team?*

Forces

- The missing of sprint goals is not easy to explain.

Solution

The x-axis of the BURNDOWN CHART represents the time, while on the y-axis the open tasks in the current sprint backlog are shown. Two lines show the difference between the ideal amount of tasks remaining (hypnotized by a linear work distribution) and the actual remaining tasks (see Figure B.5).

During the retrospective meeting the development team uses the BURNDOWN CHART to improve its own work style and to avoid too much work load in the end of a sprint. Especially sideline activities must be disclose to understand the amount of tasks done during the time. The assignee of a task explains why a task was done in the beginning or the end and shows concerns which came with the task. Also time which was spent for training and the self-teaching of new competencies should be discussed.

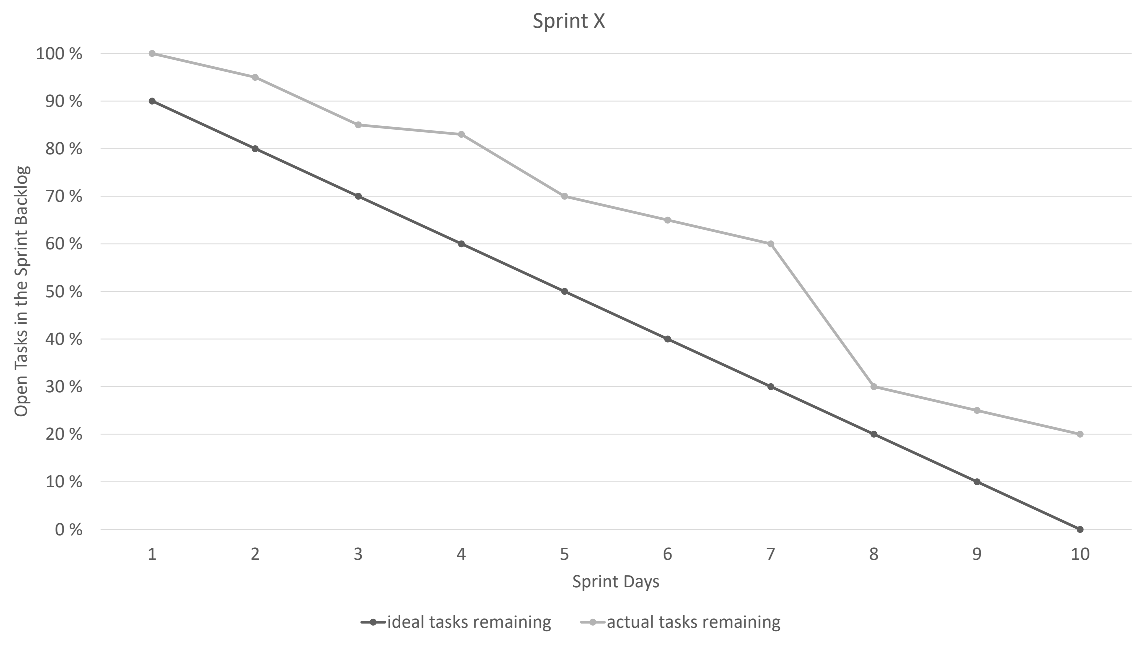

Figure B.5.: Example BURNDOWN CHART for sprint X

Variants

Development teams can also use BURNDOWN CHARTs from several sprints in the past to compare their workload.

Burnup charts are the opposite and can be used the same way.

Consequences

Benefits:

- Transparency for team, product owner and scrum master about the distribution of work load during one sprint.

Liabilities: *None known.*

Bibliography

[1] P. Abrahamsson, K. Conboy, and X. Wang. 'Lots done, more to do': the current state of agile systems development research. *European Journal of Information Systems*, 18(4):281–284, 2009.

[2] M. F. Abrar, M. S. Khan, S. Ali, U. Ali, M. F. Majeed, A. Ali, B. Amin, and N. Rasheed. Motivators for Large-Scale Agile Adoption From Management Perspective: A Systematic Literature Review. *IEEE Access*, 7:22660–22674, 2019.

[3] C. Alexander. *The Timeless Way of Building*. Oxford University Press, Oxford, United Kingdom, 1979. Google-Books-ID: H6CE9hlbO8sC.

[4] S. W. Ambler and M. Lines. *Disciplined agile delivery: A practitioner's guide to agile software delivery in the enterprise*. IBM Press, Indianapolis, Indiana, United States of America, 2012.

[5] E. Andreassen. *Inter-team Coordination in Large-scale Agile Software Development*. PhD thesis, Norwegian University of Science and Technology, Trondheim, Norway, 2015.

[6] M. A. Awad. *A Comparison between Agile and Traditional Software Development Methodologies*. PhD thesis, School of Computer Science and software Engineering, The University of Western Australia, Perth, Australia, 2005.

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

[8] K. Beck. *Implementation patterns (A Kent Beck signature book)*. Addison-Wesley Professional, Upper Saddle River, New Jersey, United States of America, 2008.

[9] K. Beck, J. Highsmith, B. Mike, A. Hunt, J. Ron, V. B. Arie, C. Ward, F. Martin, G. James, C. Alistair, F. Martin, R. Jeffries, D. Thomas, R. C. Martin, K. Schwaber, S. Mellor, and J. Sutherland. Manifesto for Agile Software Development, 2001. `https://agilemanifesto.org/` (accessed 2019-05-20).

[10] L. Briand, B. Basili, and W. Thomas. A Pattern Recognition Approach for Software Engineering Data Analysis. *IEEE Transaction on Software Engineering*, 18(11):12, 1992.

[11] B. Bruegge and A. H. Dutoit. *Object-Oriented Software Engineering. Using UML, Patterns, and Java*, volume 5. Prentice Hall, Upper Saddle River, New Jersey, United States of America, 2009.

[12] S. Buckl, F. Matthes, A. W. Schneider, and C. M. Schweda. Pattern-based Design Research in Enterprise Architecture Management. In E. Bayro-Corrochano and E. Hancock, editors, *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, volume 8827, pages 30–42. Springer International Publishing, Basel, Switzerland, 2013.

[13] M. Cohn. Cultivate Communities of Practice. In *Succeeding with Agile*. Addison-Wesley Professional, Boston, Massachusetts, United States of America, 1. edition, 2009.

[14] M. Cohn. Scaling the Product Owner. In *Succeeding with Agile*. Addison-Wesley Professional, Boston, Massachusetts, United States of America, 2009.

[15] S. Comella-Dorda, S. Lohiya, and G. Speksnijder. An operating model for company-wide agile development, 2016. `https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/an-operating-model-for-company-wide-agile-development` (accessed 2019-07-23).

[16] J. Coplien. *Software Patterns*. SIGS Books & Multimedia, New York, New York, United States of America, 1996.

[17] J. Coplien and N. Harrison. *Organizational Patterns of Agile Software Development*. Prentice Hall, Upper Saddle River, New Jersey, United States of America, 2004.

[18] K. Dikert, M. Paasivaara, and C. Lassenius. Challenges and success factors for large-scale agile transformations: A systematic literature review. *Journal of Systems and Software*, 119:87–108, 2016.

[19] T. Dingsøyr, T. E. Fægri, and J. Itkonen. What Is Large in Large-Scale? A Taxonomy of Scale for Agile Software Development. In A. Jedlitschka, P. Kuvaja, M. Kuhrmann, T. Männistö, J. Münch, and M. Raatikainen, editors, *Product-Focused Software Process Improvement*, volume 8892, pages 273–276. Springer International Publishing, Basel, Switzerland, 2014.

[20] T. Dingsøyr, M. Mikalsen, A. Solem, and K. Vestues. Learning in the Large - An Exploratory Study of Retrospectives in Large-Scale Agile Development. In J. Garbajosa, X. Wang, and A. Aguiar, editors, *Agile Processes in Software Engineering and Extreme Programming*, volume 314, pages 191–198. Springer International Publishing, Basel, Switzerland, 2018.

[21] T. Dingsøyr and N. B. Moe. Research challenges in large-scale agile software development. *ACM SIGSOFT Software Engineering Notes*, 38(5):38–40, 2013.

[22] T. Dingsøyr, S. Nerur, V. Balijepally, and N. B. Moe. A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, 85(6):1213–1221, 2012.

[23] T. Dybå and T. Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9-10):833–859, 2008.

[24] L. Ekas and S. Will. *Being Agile: Eleven Breakthrough Techniques to Keep You from "Waterfall Backward"*. IBM Press, Indianapolis, Indiana, United States of America, 2014.

[25] A. Elssamadisy. *Agile Adoption Patterns: A Roadmap to Organizational Success*. Addison-Wesley Professional, Boston, Massachusetts, United States of America, 2008.

[26] B. Flyvbjerg. Five Misunderstandings About Case-Study Research. *Qualitative Inquiry*, 12(2):219–245, 2006.

[27] C. Fuchs and T. Hess. Becoming Agile in the Digital Transformation: The Process of a Large-Scale Agile Transformation. In *Thirty Ninth International Conference on Information Systems*, volume 39, page 18, San Francisco, California, United States of America, 2018. Association for Information Systems.

[28] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, Boston, Massachusetts, United States of America, 1995.

[29] R. K. Greenleaf. *Servant Leadership: A Journey Into the Nature of Legitimate Power and Greatness*. Paulist Press, Mahwah, New Jersey, United States of America, 2002. Google-Books-ID: AfjUgMJlDK4C.

[30] N.-M. Harders. Identifying recurring Challenges and Best Practices of Agile Coaches and Scrum Masters and Documenting them as a part of a Large-Scale Agile Development Pattern Language. Master's thesis, Technical University Munich, Munich, Germany, 2019.

[31] N. Harrison. Advanced Pattern Writing. In *Proceedings of the 8th European Conference on Pattern Languages of Programms EuroPLoP '03*, Irsee, Germany, 2003.

[32] J. Heidenberg, M. Matinlassi, M. Pikkarainen, P. Hirkman, and J. Partanen. Systematic piloting of agile methods in the large: two cases in embedded systems development. In *International Conference on Product Focused Software Process Improvement*, pages 47–61, Basel, Switzerland, 2010. Springer International Publishing.

[33] A. R. Hevner, S. T. March, J. Park, and S. Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):76–106, 2004.

[34] J. A. Highsmith and J. Highsmith. *Agile Software Development Ecosystems*. Addison-Wesley Professional, Boston, Massachusetts, United States of America, 2002.

[35] M. Kalenda, P. Hyna, and B. Rossi. Scaling agile in large organizations: Practices, challenges, and success factors. *Journal of Software: Evolution and Process*, 2018.

[36] D. Karlstrom and P. Runeson. Combining Agile Methods with Stage-Gate Project Management. *IEEE Software*, 22(3):43–49, 2005.

[37] K. Kuusinen, P. Gregory, H. Sharp, L. Barroca, K. Taylor, and L. Wood. Knowledge Sharing in a Large Agile Organisation: A Survey Study. *Agile processes in software engineering and extreme programming*, pages 135–150, 2017.

[38] C. Larman and B. Vodde. *Large-scale scrum: More with LeSS*. Addison-Wesley Professional, Boston, Massachusetts, United States of America, 2016.

[39] D. Leffingwell. Scaled Agile Framework – SAFe for Lean Enterprises, 2019. `https://www.scaledagileframework.com/` (accessed 2019-07-22).

[40] T. C. Lethbridge, S. E. Sim, and J. Singer. Studying Software Engineers: Data Collection Techniques for Software Field Studies. *Empirical Software Engineering*, 10(3):311–341, 2005.

[41] H. Merisalo-Rantanen, T. Tuunanen, and M. Rossi. Is Extreme Programming Just Old Wine in New Bottles: A Comparison of Two Cases. *Journal of Database Management*, 16(4):41–61, 2005.

[42] G. Meszaros and J. Doble. A Pattern Language for Pattern Writing. *Proceedings of International Conference on Pattern languages of program design*, 3:529 – 574, 1997.

[43] N. B. Moe, T. Dingsøyr, and K. Rolland. To schedule or not to schedule? An investigation of meetings as an inter-team coordination mechanism in large-scale agile software development. *IJISPM - International Journal of Information Systems and Project Management*, 6(3):45–59, 2018.

[44] N. B. Moe, D. Šmite, A. Šāblis, A.-L. Börjesson, and P. Andréasson. 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*, pages 1–8, Torino, Italy, 2014. ACM Press.

[45] I. F. Oskam. T-shaped engineers for interdisciplinary innovation: an attractive perspective for young people as well as a must for innovative organisations. In *37th Annual Conference - Attracting students in Engineering*, page 11, 2009.

[46] M. Paasivaara. Adopting SAFe to Scale Agile in a Globally Distributed Organization. In *2017 IEEE 12th International Conference on Global Software Engineering (ICGSE)*, pages 36–40, Buenos Aires, Argentina, 2017. IEEE.

[47] M. Paasivaara and C. Lassenius. Communities of practice in a large distributed agile software development organization – Case Ericsson. *Information and Software Technology*, 56(12):1556–1577, 2014.

[48] M. Paasivaara and C. Lassenius. Scaling Scrum in a Large Globally Distributed Organization: A Case Study. In *2016 IEEE 11th International Conference on Global Software Engineering (ICGSE)*, pages 74–83, Orange County, California, USA, 2016. IEEE.

[49] G. Papadopoulos. Moving from Traditional to Agile Software Development Methodologies Also on Large, Distributed Projects. *Procedia - Social and Behavioral Sciences*, 175:455–463, 2015.

[50] R. Phaal and G. Muller. Towards Visual Strategy: An Architectural Framework for Roadmapping. In *PICMET '07 - 2007 Portland International Conference on Management of Engineering & Technology*, pages 1584–1592, Portland, Oregon, United States of America, 2007. IEEE.

[51] R. Pichler and S. Roock. *Agile Entwicklungspraktiken mit Scrum*. dpunkt. verlag, Heidelberg, Germany, 2011.

[52] D. Riehle. Design pattern density defined. *SAP Research*, page 12, 2009.

[53] C. Robson. *Real world research: a resource for social scientists and practitioner-researchers*. Blackwell Publishers, Oxford, United Kingdom, 2nd ed edition, 2002.

[54] K. Rolland, V. Mikkelsen, and A. Næss. Tailoring Agile in the Large: Experience and Reflections from a Large-Scale Agile Software Development Project. *Agile processes, in software engineering, and extreme programming*, pages 244–251, 2016.

[55] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

[56] Scaled Agile Inc. Product Owner – Scaled Agile Framework, 2019. `https://www.scaledagileframework.com/product-owner/` (accessed 2019-07-23).

[57] A. Scheerer. *Coordination in Large-Scale Agile Software Development*. Progress in IS. Springer International Publishing, Basel, Switzerland, 2017.

[58] A. Scheerer, T. Hildenbrand, and T. Kude. Coordination in Large-Scale Agile Software Development: A Multiteam Systems Perspective. In *2014 47th Hawaii International Conference on System Sciences*, pages 4780–4788, 2014.

[59] K. Schwaber. Online Nexus Guide | Scrum.org, 2019. `https://www.scrum.org/index.php/resources/online-nexus-guide` (accessed 2019-06-21).

[60] K. Schwaber and M. Beedle. *Agile software development with Scrum*. Prentice Hall, Upper Saddle River, New Jersey, United States of America, 1 edition, 2002.

[61] Scrum.org. What is Scrum?, 2019. `https://www.scrum.org/resources/what-is-scrum` (accessed 2019-06-21).

[62] Scrum@Scale LLC. Scrum at Scale, 2019. `https://www.scrumatscale.com/scrum-at-scale-guide/` (accessed 2019-06-21).

[63] H. Svensson and M. Host. Introducing an Agile Process in a Software Maintenance and Evolution Organization. In *Ninth European Conference on Software Maintenance and Reengineering*, pages 256–264, Manchester, United Kingdom, 2005. IEEE.

[64] The LeSS Company B.V. Product Owner - Large Scale Scrum (LeSS), 2019. `https://less.works/less/framework/product-owner.html` (accessed 2019-07-23).

[65] Ö. Uludağ. Large-Scale Agile Development Pattern Graph, 2019. `https://scaling-agile-hub.sebis.in.tum.de/#/patterns` (accessed 2019-07-11).

[66] Ö. Uludağ, N.-M. Harders, and F. Matthes. Documenting Recurring Concerns and Patterns in Large-Scale Agile Development. *Association for Computing Machinery*, 1(1):15, 2019.

[67] Ö. Uludağ, M. Kleehaus, C. Caprano, and F. Matthes. Identifying and Structuring Challenges in Large-Scale Agile Development Based on a Structured Literature Review. In *2018 IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC)*, pages 191–197, Stockholm, Sweden, 2018. IEEE.

[68] Ö. Uludağ, M. Kleehaus, N. Dreymann, C. Kabelin, and F. Matthes. Investigating the Adoption and Application of Large Scale Scrum at a German Automotive Manufacturer. In *Proceedings of the 14th International Conference on Global Software Engineering*, Montreal, Quebec, Canada, 2019. IEEE.

[69] Ö. Uludağ, M. Kleehaus, X. Xu, and F. Matthes. Investigating the Role of Architects in Scaling Agile Frameworks. In *2017 IEEE 21st International Enterprise Distributed Object Computing Conference (EDOC)*, pages 123–132, Quebec City, Quebec, Canada, 2017.

[70] Ö. Uludağ and F. Matthes. Identifying and Documenting Recurring Concerns and Best Practices of Agile Coaches and Scrum Masters in Large-Scale Agile Development. *HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 26*, page 22, 2019.

[71] Ö. Uludağ, N. Reiter, and F. Matthes. Improving the Collaboration Between Enterprise Architects and Agile Teams - A Multiple-Case Study. *Architecting the Digital Transformation, Springer-Verlag*, 2020.

[72] T. Wellhausen and A. Fiesser. How to write a pattern?: a rough guide for first-time pattern authors. In *Proceedings of the 16th European Conference on Pattern Languages of Programs - EuroPLoP '11*, pages 1–9, Irsee, Germany, 2011. ACM Press.

[73] E. Wenger, R. A. McDermott, and W. Snyder. *Cultivating Communities of Practice: A Guide to Managing Knowledge*. Harvard Business School Press, Boston, Massachusetts, United States of America, 2002. Google-Books-ID: m1xZuNq9RygC.

[74] R. J. Winter. Agile Software Development: Principles, Patterns, and Practices: Robert C. Martin with contributions by James W. Newkirk and Robert S. Koss. *Performance Improvement*, 53(4):43–46, 2014.

[75] D. A. Wintersteiger. *Scrum: Schnelleinstieg*. entwickler.press, Frankfurt am Main, Germany, 4. edition, 2015.