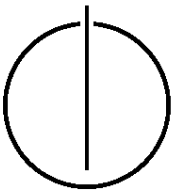
DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Information Systems

Empirical Studies to Identify Coordination- and Methodology Patterns in Large-Scale Agile Development

Moritz Schüll



DEPARTMENT OF INFORMATICS

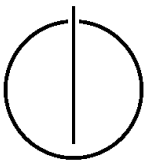
TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Information Systems

Empirical Studies to Identify Coordination- and Methodology Patterns in Large-Scale Agile Development

Empirische Studien zur Identifikation von Koordinations- und Methodenmustern bei der skalierten agilen Entwicklung

Author:	Moritz Schüll
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Ömer Uludağ, M. Sc.
Submission Date:	August 15, 2019



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

Munich, August 15, 2019 Moritz Schüll

Abstract

Agile software development methodologies have been designed for use in small teams. The objective of agile methods is to increase the flexibility of the development teams, to adapt to changing requirements by closely working together with the customer, and to produce products that actually satisfy the user in the end. During the past years, due to the shown benefits of agile methodologies, larger companies have been increasingly interested in applying them in their software development projects as well. Using agile methodologies in large-scale agile software development projects with multiple teams, however, creates additional concerns for companies, because those methodologies have not been designed with multiple teams and locations in mind. Concerns such as not working co-located, increased need for communication, and dependencies between the different teams are not addressed by standard agile methodologies. New good practices are required to apply agile methods successfully at scale.

So far, this area of concerns and new practices for large-scale agile software development is not broadly covered in research. Therefore, the Chair of Software Engineering for Business Information Systems (sebis chair) at the Technical University of Munich has developed a Large-Scale Agile Development Pattern Language. This pattern language will contain concerns of various stakeholders and patterns that address those concerns. This thesis will contribute to filling this pattern language with concerns and pattern candidates observed in practice, by conducting an empirical study at a large German software vendor. The thesis looks at various stakeholders of the large-scale agile development program and seeks to identify their recurring concerns in coordination and application of agile methods at large scale. Further, good practices that are employed by the stakeholders to overcome their concerns are collected and documented in a formalized way, according to the pattern language. This thesis is part of a larger research project. The collected concerns and good practices, together with the likes being collected at other projects, contribute to this larger project. Its goal is to create a collection of patterns that is designed to help practitioners to successfully apply agile methodologies at large-scale projects.

Contents

Outline of the Thesis

Chapter 1: Introduction

The first chapter presents the motivation of the thesis. It explains why research about applying agile methodologies in large-scale projects is worth pursuing and how the objectives of this thesis are connected to the current state of research and practice. The end of the chapter outlines the approach chosen to accomplish the objectives.

Chapter 2: Foundations

The Foundations define important terms and present the context in which this thesis is placed. The chapter summarizes existing research by the sebis chair that this thesis builds on. Also, the theoretical foundations of agile methods, pattern-based solution approaches, and coordination theory are presented.

Chapter 3: Related Work

The third chapter compares several pattern languages in the agile and large-scale context. Further, it discusses related work on coordination in large-scale agile software development.

Chapter 4: Identifying Recurring Concerns and Practices

This chapter presents the empirical study that is conducted in the thesis. First, the scientific methodology of the empirical study is explained. Then, the concerns and practices that are found throughout the course of the study are discussed.

Chapter 5: Discussion

The Discussion outlines the key findings of the thesis and reviews the research quality of the presented study. Limitations of the work are discussed.

Chapter 6: Conclusion

The final chapter summarizes the work and presents an outlook to potential future work.

1. Introduction

1.1. Motivation

For a long time, methodologies like the Waterfall Model and the Spiral Model have been largely used for software development [12, 58]. These traditional software development methodologies, however, fail to adapt to changes in requirements and project scope during development; they rely on comprehensive upfront planning and do not include frequent interaction with the project customers. This approach to structure the software development as a defined process leads to inability to react promptly to unexpected events during the development [65].
Form the 1990s until the early 2000s, several lightweight software development methodologies have emerged that make software development more adaptive to changing requirements and reduce time-to-market. In 1995, Scrum was presented by Ken Schwaber and Jeff Sutherland at the Object-Oriented Programming, Systems, Languages & Applications (OOPSLA) conference [65], and in 1999 Kent Beck published Extreme Programming (XP) [4]. In 2001, the Agile Manifesto assembled the values of these and other methodologies known under the term "Agile Methodologies" [5]. Agile methodologies approach software development as an empirical process [65]. They focus on adaption to change, learning about the project and customer needs during development, iterative work, and incremental product building [29]. They are designed for small development teams that work co-located and do small, frequent releases [13].
By today, agile methodologies have become very popular in the software development world, with Scrum and XP being among the most commonly used [40]. Due to their shown benefits, agile methodologies have become increasingly interesting for larger companies [23]. However, agile methodologies are difficult to introduce in larger projects. Applying agile in large projects brings new concerns that are not addressed by the normal agile practices, like inter-team coordination, dealing with organizational structures, and additional stakeholders [23].
At the International Conference on Agile Software Development (XP Conference) in 2010, practitioners voted "agile and large projects" to be their most pressing question they want research to be conducted on [33]. To fill this gap between research topics and practitioners' demands, the Chair of Software Engineering for Business Information Systems (sebis chair) of the Technical University of Munich created a Large-Scale Agile Development Pattern Language (LSADPL), which will collect and formalize concerns and good practices to address them found in the industry [74].
At the XP Conference in 2013, participants were asked to state what they see as the most

important research topics in large-scale agile software development [25]. They ranked "inter-team coordination" as a top priority research topic [25]. This thesis tries to address this research topic by conducting an empirical study at a large German software vendor, focusing on concerns and practices in

- coordination of work and
- applying the methodology

in large-scale agile software development. The concerns and practices identified in the study will be structured according to the LSADPL of the sebis chair.

1.2. Research Objectives

Following the mentioned research motivation, this bachelor's thesis seeks to contribute to the areas of coordination and methodology in large-scale agile development. To meet this objective, we formulate three research questions (RQs) and answer them in this work.
RQ1. What are recurring coordination and methodology concerns in large-scale agile development? The objective of the first research question is to identify concerns of the categories coordination and methodology. The goal is to find concerns in the literature and from practice. Concerns are collected and validated at the industry partner's organization. To identify recurring concerns, the aim is to collect multiple opinions on each concern.
RQ2. What are good practices for addressing recurring coordination and methodology concerns in large-scale agile development?
The goal of the second research question is to identify good practices that are used at the case company to address and solve concerns. This research question is building on the concerns that are identified in research question 1.
RQ3. Which anti-patterns regarding coordination and methodologies should be avoided in large-scale agile development?
In contrast to the second research question, the third one focuses on bad practices that the case organization tried to employ in order to solve the identified concerns and that turned out not to work in the expected way or even worsened the situation.

1.3. Approach

This thesis is structured following the Pattern-Based Design Research (PDR) approach by Buckl et al. [15] as shown in Figure 1.1. Accordingly, the desired outcome of the thesis are artifacts, in the form of pattern candidates, that are relevant and rigorous. The work is based on an existing knowledge base consisting of the LSADPL by the sebis chair [74] and a list of concerns that have already been identified and categorized in literature. By building the research artifacts on the existing knowledge base rigor is ensured [15, 39]. To identify further concerns and good practices to address the concerns, a single-case, embedded study is conducted at a large German software vendor [80]. This ensures relevancy of

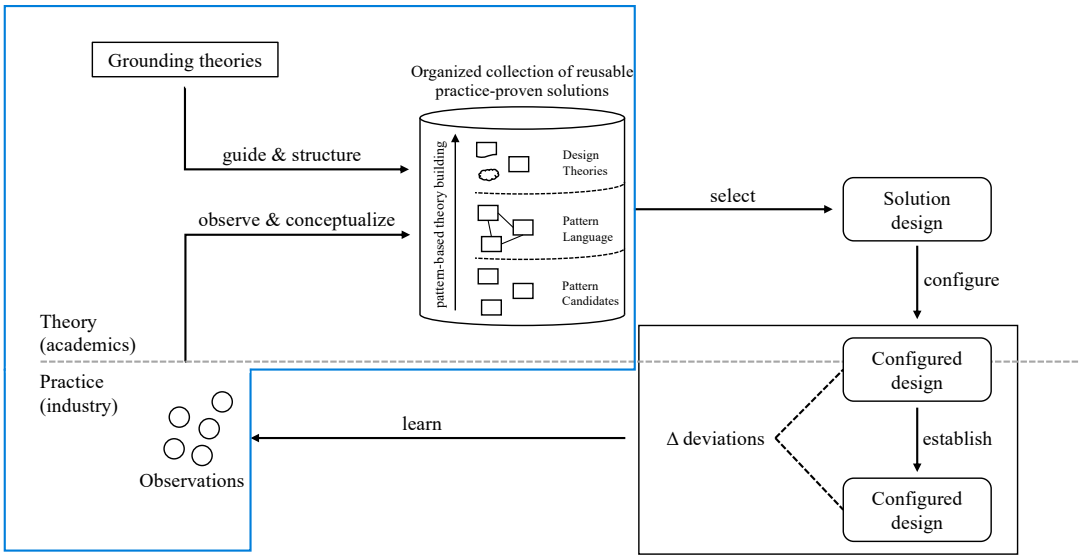


Figure 1.1.: Overview of the approach of this thesis based on [15]

the created research artifacts for practitioners [39].

To select the appropriate research strategy for the study at the industry partner, Yin [80] and Benbasat et al. [8] formulate three respectively four questions that can be used as a guideline. These questions regard whether the observed phenomenon can be studied outside its natural setting, whether the study focuses on contemporary events, whether behavioral control is required, and whether there is already an established theoretical base covering the phenomenon of interest. As this thesis focuses on finding good practices from industry, the phenomenon of interest cannot be observed outside of its natural setting. The focus is on agile methodologies at scale, which is a novel field of research, so the study focuses on contemporary events and there is no sound theoretical base established yet [23, 62]. Finally, the phenomenon is observed in a productive organizational environment, so behavioral control is not indicated. Considering these points and the guidance in [80], we selected a single-case, embedded study research method. The research can be classified as descriptive, because it describes the practices the case organization is employing to address concerns it faces [60]. It can also be classified as exploratory to a certain extent, because it seeks new insights about concerns in large-scale agile development. According to the PDR developed by Buckl et al. [15], the work of this thesis can be mapped to the first two steps *observe & conceptualize* and *pattern-based theory building & nexus instantiation*. Figure 1.1 highlights the two phases of the PDR approach that are implemented in this thesis.

We selected first, second, and third degree types of data collection for this research, comprised of interviews, meeting observations, and other work artifacts like documentation and coaching slides [60]. The insights gained from the software company are documented

as artifacts according to the LSADPL [74]. For good or bad practices to be considered a pattern or anti-pattern, the pattern language uses the *rule of three* [20]. It states that a practice has to be observed in three independent organizations to be considered a pattern. Thus, because the conducted study covers a single case, the artifacts created in this work are not considered to be finalized patterns but pattern candidates.

The remainder of this bachelor's thesis is structured as follows. In Section 2, the thesis first explains the relevant scientific foundations, which are necessary to understand the following work. Section 3 takes a look at other research conducted in the area of scaling agile practices and coordination of work in large-scale development. Then, in Section 4 we present the study at the German software vendor. The section explains the research methodology and the findings in detail. Section 5 presents the key findings of this work and discusses research quality and limitations. In the final Section 6, we summarize the work and present possible future work.

2. Foundations

This chapter gives an overview of the foundations on which this thesis is built. Relevant terms are defined and concepts that are used throughout the following work are explained. First, in Section 2.1 the theory and origins of patterns are explained. It includes a description of how patterns differ from good practices and a section about how patterns should be written. Second, in Section 2.2 agile software development is defined and the values behind it are presented. It is also discussed how the agile approach differentiates from other methodologies for software projects. Third, in Section 2.3 large scale agile development is described. Concerns of applying agile methodologies at scale that have been identified in literature are shown, as well as the Large-Scale Agile Development Language (LSADPL) that this thesis is building on. Finally, in Section 2.4 the theoretical background on work coordination in organizations is discussed.

2.1. Patterns

This section provides the theoretical background and definitions for the concepts of patterns and pattern languages that are used throughout this thesis.

2.1.1. Definitions

Patterns in the setting of software development have their origin in Christopher Alexander's work on patterns in urban design and construction [1, 20]. Alexander created the concept of patterns and pattern languages to document solutions to recurring problems that can be applied multiple times. His definition of a pattern is as follows:

> *Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.*
>
> — Christopher Alexander, A Pattern Language [1]

Most work in software engineering related to patterns and pattern languages reference to Alexander's work [20]. As patterns are defined as solutions to recurring problems, they are not "invented" and do not create new knowledge [45]. Rather, they are emergent [45]. This means they capture solutions from practice and compile knowledge in a formal way, such that the knowledge about the solution can be easier accessed by a wider range of people [20, 45]. The solution is not captured completely in detail as it is observed, but rather

in a suitable abstraction to make the solution applicable in a broader context [20].
Besides the concept of patterns there is also the concept of Anti-Patterns. Anti-Patterns describe alleged solutions to a recurring problem, but lead to more negative consequences than positive ones [47]. Additionally, Anti-Patterns contain a revised solution that describes the changes that have to be made to turn the Anti-Pattern into a beneficial pattern [47, 74].
Usually, patterns do not come individually but as part of a pattern language [21]. This is due to the fact that applying individual patterns does not suffice to build complex real-world systems or organizations [64]. The usage of one pattern often yields the opportunity or need for the usage of other patterns [45]. Pattern languages connect different patterns with each other and help to understand the collection of patterns as a whole [1]. Considering the whole language, a complete system or organization can be created [21]. Thus, a pattern language can be described as a collection of inter-linked patterns from a common area of interest [45]. Pattern languages help the reader to identify suitable patterns for their situation [47]. When several patterns of one pattern language are applied together or after each other, this is called a pattern *sequence* [21, 45].
As already pointed out in the introduction, this thesis seeks to identify and document good practices. The good practices are also called *pattern candidates* within this work. The *rule of three* [20] is used by the LSADPL to determine the difference of a pattern candidate an actual pattern [74]. A pattern has to be observed and documented at three different, independent organizations; a pattern candidate has to be observed in practice at least once.

2.1.2. Writing Patterns

As stated in the previous section, patterns are not invented by an author, but are emergent. However, there are certain guidelines on how to formulate a pattern properly to make it as useful as possible to the audience. The European Conference on Pattern Languages of Programs (EuroPLoP) provides a website with an introductory package of four papers[1] to get started with pattern writing. Other authors like Coplien [20] and Kelly [45] also provide advice on how to write good patterns. The name, problem, and solution sections are most likely to be the first ones a potential user will look at [36, 45, 78]. Consequently, when starting to write a pattern, Kelly [45] and Meszaros and Doble [51] recommend to begin with these three sections of the pattern. Similarly, Harrison [36] and Wellhausen and Fießer [78] suggest to start with the solution and problem sections. The context, forces, and consequences are further sections identified to be important for patterns [36, 45, 51, 78]. Kelly [45] and Harrison [36] suggest the **name** of the pattern being based on the *thing* the pattern is building and thus being comprised of nouns. Kelly [45] further underlines to avoid verb-based naming. Meszaros and Doble [51] present a noun based naming-scheme as well. However, they also mention the possibility to name the pattern according to the process used to create the solution using verb phrases [51]. Whether being a noun or verb

[1] europlop.net Pattern Writing: https://www.europlop.net/content/start-writing
The papers are [36, 38, 51, 78].

phrase, the bottom line is that the chosen pattern name should be evocative and conjure up images that convey an idea of the solution to the reader [36, 51]. The **problem** section of the pattern should not be phrased like a "How do you do X?" question. This tends to invoke a "Do X" solution [78]. Kelly [45] describes that during the writing process information might move back and forth between the problem and forces sections. The **forces** of a pattern help the reader understand the significance of the problem and why the problem is actually difficult to solve [45]. Both, Kelly [45] and Wellhausen and Fießer [78] emphasize to match the forces with the **consequences**. Each consequence should describe how one force is being solved by the solution of the pattern. The **solution** section should be sufficiently abstract to make the pattern applicable to different users, but specific enough to guide the reader what to do [20, 45]. Again, Kelly [45] describes that information might move back and forth between the solution and consequences sections during the writing process. Finally, the **context** describes the situation in which the pattern is applicable [20]. The context might contain a history of patterns that have already been applied and defines a scope in which the pattern can be used [20]. Wellhausen and Fießer [78] stress that the context is not modified by the application of the pattern. The process of writing a pattern itself is iterative [38, 45, 78]. During the writing process the author will most likely rethink and sharpen the idea of what the pattern is describing [45]. All the sections should be revisited after the first version of the pattern is created [78].

2.2. Agile Software Development

After the concepts of patterns and pattern languages have been clarify, this section continues with explaining agile software development. Agile software development is an important part of the foundations of this thesis.

2.2.1. Definitions and Values

The core assumption that is common for all agile methodologies is the recognition of the software development process as being empirical [79]. Industrial processes can be classified either as defined or empirical [65, 79]. A defined process is a process that can be clearly defined upfront. Every time it is executed, it produces the same output for a given input [79]. An empirical process, however, is a process that cannot be designed in its entirety before execution. It has to deal with uncertainties and requires frequent investigations of changes and adaptions to new demands [79]. Agile software development methodologies have been designed to deal with those properties of an empirical process and are able to handle changing requirements and uncertainty [79]. In 2001, the 'Agile Manifesto' [5] was created to condense the objectives of agile methodologies into four values and twelve principles. These values are [5]:

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

The authors of the Agile Manifesto emphasize that "while there is value in the items on the right, we value the items on the left more" [5]. This means that individuals and interactions, working software, customer collaboration, and responding to change are more important than processes and tools, documentation, contracts, and following a plan.

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

Table 2.1.: The 12 principles of the Agile Manifesto [5]

The 12 principles of the Agile Manifesto can be found in Table 2.1. The values and principles that are presented by the Agile Manifesto clearly reflect the recognition of software development as an empirical process. They emphasize to collaborate with the customer of the product and to adapt to changing environments and needs, instead of sticking to an upfront designed, defined process.

2.2.2. Distinction from other Methodologies

Besides agile methodologies, which are following an empirical process control (as explained in the previous section), there are also other methodologies for software development process management that are following defined process control structures [65].

These include sequential methodologies, like the Waterfall Model [58] and the V-Model [61] which are structured along milestones and always focus on one activity at a time. And also iterative methodologies like the Spiral Model [12] and the Unified Process [41] are mostly following defined process control. These "traditional" methodologies for software development guide people by standardized processes and organization. In contrast, agile methodologies are designed to tailor processes to the teams and particular projects [18]. When sequential models are applied, all requirements have to be defined upfront the project in the design phase. No adaption is possible throughout project execution [65]. Iterative methods like the Spiral Model, while at least being responsive to change between activities, still assume that issues of a finished activity cannot be changed or altered afterwards [12]. Agile methodologies address these shortcomings by time-boxing development iterations and frequently incorporating customer feedback into the development [79].

2.2.3. Scrum

As already mentioned in the introduction, there are several different frameworks for agile methodologies. According to the annual "State of Agile" report, Scrum and hybrids of Scrum and XP are among the most commonly used agile methods at the moment [40]. The Scrum methodology was used in all observed teams at the case organization, therefore this section gives an overview of Scrum.

The Scrum framework was presented in 1995 by Ken Schwaber and Jeff Sutherland at the OOPSLA conference [65]. It treats systems development processes "as a controlled black box" [65], which is how Schwaber calls empirical processes in this work. Scrum has been designed for small, cross-functional teams between three and six people [65]. Today, Sutherland and Schwaber are maintaining "The Scrum Guide" [67] which holds up to date information about Scrum and its execution.

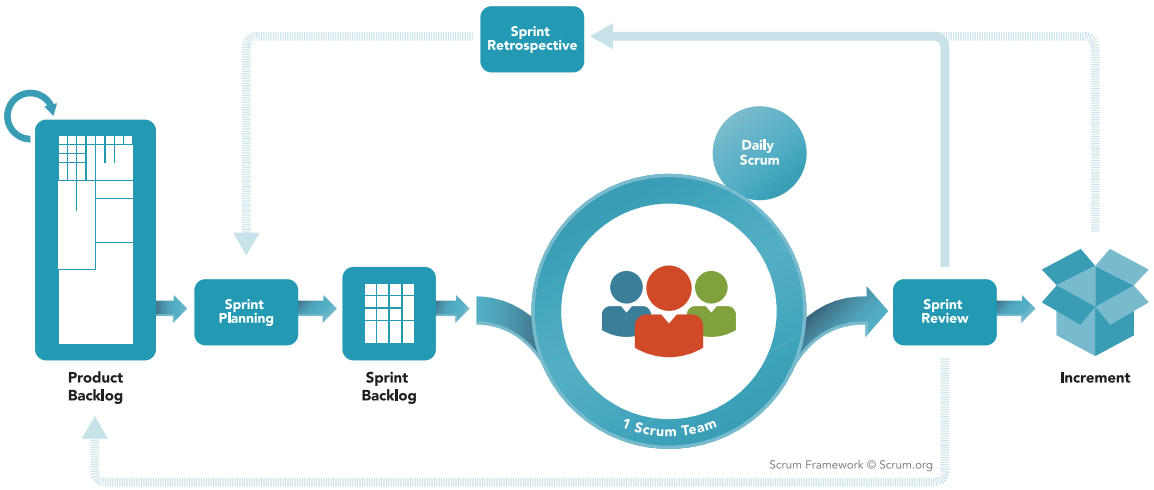


Figure 2.1.: Scrum Framework illustration by [68]

The main concepts of Scrum are three artifacts, three roles in the Scrum team, and five events [59, 65, 67]:
Product Backlog. The product backlog is the first artifact that is created for the development [59]. It lists all features and required capabilities of the product. New work items may be added to the product backlog in the beginning and also throughout the course of development. The items in the product backlog are prioritized by the customer [67].
Sprint Backlog. The sprint backlog is the artifact that contains items form the product backlog on which the team is working during the current iteration [67]. The items from the product backlog are selected according to their priority [59]. The sprint backlog must not be altered during an active sprint execution.
Increment. An increment is the output artifact produced by the development team during one iteration. It is the result of all sprint backlog items that were finished in the current sprint [67].
Product Owner. The role of the Product Owner (PO) is twofold [59]. The PO has to ensure that the needs and priorities of the customer are understood and grooms and prioritizes the product backlog. The PO is also responsible for communicating to the development team what has to be built and for ensuring that the project is economically viable. "[...] the product owner is focused on building the right product" [59].
Scrum Master. "The Scrum Master is a servant-leader for the team" [67]. The Scrum Master ensures that everybody in the team understands Scrum, its values and principles, and that it is applied correctly.
Development Team. The development team is doing the work that is defined in the sprint backlog. They are producing a product increment in each iteration. Ideally, the development team has all the required skills to build the required outcome of the project. "[...] the development team is focused on building the product right" [59].
Sprint. An iteration in Scrum is called sprint. A sprint is a time-boxed period of time. During a sprint the development team creates a new product increment based on the sprint backlog and the results of the sprint planning meeting [59].
Sprint Planning. A sprint is preceded by a sprint planning meeting. The meeting is time-boxed and the Scrum Master enforces this time-limit. The complete scrum team takes part in the planning and determines collectively what should be part of the sprint backlog for the next product increment [67].
Daily Scrum. Every day during a sprint, the development team gathers for a short daily scrum meeting. The team inspects the progress and adapts the work plan for the upcoming day [67].
Sprint Review. After the sprint, a sprint review meeting is used to inspect the increment that has been produced during the sprint. The team can make adjustments to the product backlog, if necessary [59].
Sprint Retrospective. After the sprint review meeting and before the next sprint planning meeting, the complete scrum team holds a sprint retrospective meeting. They analyze their workflows and processes to identify opportunities to improve work and collaboration [67]. It is important for continuous and incremental learning [59].

As these artifacts, events, and roles reflect, the goal of Scrum is to iteratively create product increments. Because the sprints are short and time-boxed, a new increment is created very frequently and valuable customer feedback can be gathered based on it. This allows the Scrum team to quickly adjust to feedback, new requirements, or changes in the environment. This helps to avoid developing the "wrong" product [59].

2.3. Large-Scale Agile Development

This thesis builds on the results of previous research that has been conducted in the context of the same research project by the sebis chair. The utilized knowledge base consists of a literature review that identifies recurring concerns in large-scale agile development [75] and the LSADPL [74]. This chapter explains them in the following. Additional literature sources for concerns in large-scale agile software development are discussed in the following as well.

2.3.1. Definition

To talk about large-scale agile and the benefits and concerns related to it, first the term *large-scale* has to be defined. In related literature no single final definition can be found. Both, Scheerer [62] and Uludağ et al. [74] reference an attempt by Dingsøyr and Moe [26] to collect definitions of large-scale agile development at the 2014 XP Conference. Dingsøyr and Moe [26] collected a list of 16 definitions given by participants. They recognize that many of the definitions are based on some kind of size metric, e.g., team number, number of people involved, or lines of code. However, they find that these metrics are describing the term *large-scale* very inconsistently [26]. In another paper, Dingsøyr et al. [24] propose a taxonomy that is based on the *7 +/- 2 rule of thumb*. According to this taxonomy, *large-scale* is defined as more than one team (with team size following the 7 +/- 2 rule of thumb) and less than ten teams. Projects involving more than nine teams are defined as *very large-scale* by Dingsøyr et al. [24]. The distinction made by [24] between *small-scale* and *large-scale* can be explained by the additional need for inter-team coordination as soon as more than one team are working together on a project [24]. In this thesis the taxonomy by Dingsøyr et al. [24] is used to define what is *large-scale*.

2.3.2. Concerns

Uludağ et al.: Identifying and Structuring Challenges in Large-Scale Agile Development based on a Structured Literature Review. 2018.

Using a structured literature review, Uludağ et al. [75] compiled a list of 79 concerns in large-scale agile development from 73 sources. The reviewed literature was coded following the approach by Cruzes and Dybå [22], using an individual code family for each

pattern type and the concerns. Similar code families are used in this thesis to code the collected data. The found concerns are linked to stakeholders and are categorized. The literature review identified 14 consolidated stakeholder categories and 11 concern categories. The concern categories are *culture & mindset*, *communication & coordination*, *enterprise architecture*, *geographical distribution*, *knowledge management*, *methodology*, *project management*, *quality assurance*, *requirements engineering*, *software architecture*, and *tooling*. As this thesis focuses on coordination and methodology, the categories *communication & coordination* and *methodology* are of particular interest. Of the 79 concerns identified by Uludağ et al. [75] 11 are of one of these two categories. The relevant concerns are shown in Table 2.2.

ID	Name	Category
C-1	How to coordinate multiple agile teams that work on the same product?	Communication & Coordination
C-6	How to deal with incorrect practices of agile development?	Methodology
C-16	How to establish self-organization?	Communication & Coordination
C-20	How to facilitate communication between agile teams and other teams using traditional practices?	Communication & Coordination
C-44	How to deal with communication gaps with stakeholders?	Communication & Coordination
C-49	How to deal with increased efforts by establishing inter-team communication?	Communication & Coordination
C-59	How to establish a common understanding of agile thinking and practices?	Methodology
C-63	How to explain requirements to stakeholders?	Communication & Coordination
C-75	How to form and manage autonomous teams?	Communication & Coordination
C-78	How to build an effective coaching model?	Methodology
C-79	How to synchronize sprints in the large-scale agile development program?	Communication & Coordination

Table 2.2.: The 11 concerns from [75] that are relevant to this thesis

Scheerer: Coordination in Large-Scale Agile Software Development. 2017.

Scheerer conducted a multiple-case, embedded study at SAP [62], a large German enterprise software company. The objective was to study coordination of teams in multi-team setups in large-scale agile software development. They focused on how this coordination changes if the coordination configuration changes. During the case study, five multi-team setups were examined. They created a research framework to study those multi-team setups that comprises of [62]:

- a *coordination configuration* that integrates three aspects of coordination (*type*, *locus*, and *direction*),
- a *trigger* that enacts a coordination configuration as a reaction,

- the *integrating conditions* common understanding, predictability, and accountability that lead to coordinated action,
- and the *contingency factors* like uncertainty or dependencies, that can affect the reaction of the system to a trigger and which coordination configuration is enacted.

ID	Name	Category
C-80	How to deal with a competing concept deadlock?	Communication & Coordination
C-81	How to deal with missing communication of decommitment?	Communication & Coordination
C-82	How to deal with unclear mutual expectations?	Communication & Coordination
C-83	How to deal with lacking knowledge of another team's activities?	Communication & Coordination
C-84	How to deal with unknown dependencies between teams?	Communication & Coordination
C-85	How to deal with increase in geographic dispersion?	Geographical Distribution
C-86	How to deal with corruption of a shared codebase?	Methodology
C-87	How to deal with a work item spanning across teams?	Communication & Coordination
C-88	How to deal with unclear work items?	Communication & Coordination
C-89	How to deal with unclear usage of a new development framework?	Communication & Coordination
C-90	How to deal with major testing failures of new feature?	Communication & Coordination
C-91	How to deal with new cross-team feature originating from one team?	Communication & Coordination
C-92	How to deal with priority conflict within takt?	Communication & Coordination
C-93	How to deal with assumption mismatch?	Communication & Coordination
C-94	How to deal with late delivery of needed functionality?	Communication & Coordination
C-95	How to rapidly deliver a necessary patch?	Methodology
C-96	How to deal with unresolved prioritization of a topic?	Communication & Coordination
C-97	How to deal with a cross-team item facing asymmetric team knowledge?	Communication & Coordination
C-98	How to deal with recognition of a reuse possibility?	Communication & Coordination
C-99	How to deal with discovery of redundancies?	Communication & Coordination

Table 2.3.: The 20 change processes by [62]

In the first step, each of the five observed multi-team setups were scrutinized in a single-case analysis according to the described research framework. Afterwards, integrating conditions were identified in a cross-case analysis. 20 change processes where coordination did or did not work well in the different multi-team setups are presented. Scheerer [62] derived coordination configurations from these change processes and compared them. The 20 change processes are displayed in Table 2.3. The findings of the study show that if a

trigger event in the coordination between teams reveals an insufficient realization of an integration condition, a change in coordination configuration is invoked that leads to achieve the missing integration conditions [62]. Further, Scheerer [62] identified six specific coordination configurations that realize one of the integration conditions.

2.3.3. Large-Scale Agile Development Pattern Language

This thesis uses the LSADPL as a basis [74]. The pattern candidates and concerns that are observed in the study with the industry partner are formatted and categorized according to the structures of the LSADPL. This section describes the LSADPL and its important concepts. A comparison of the LSADPL to other relevant pattern languages from the field of agile development can be found in Section 3.

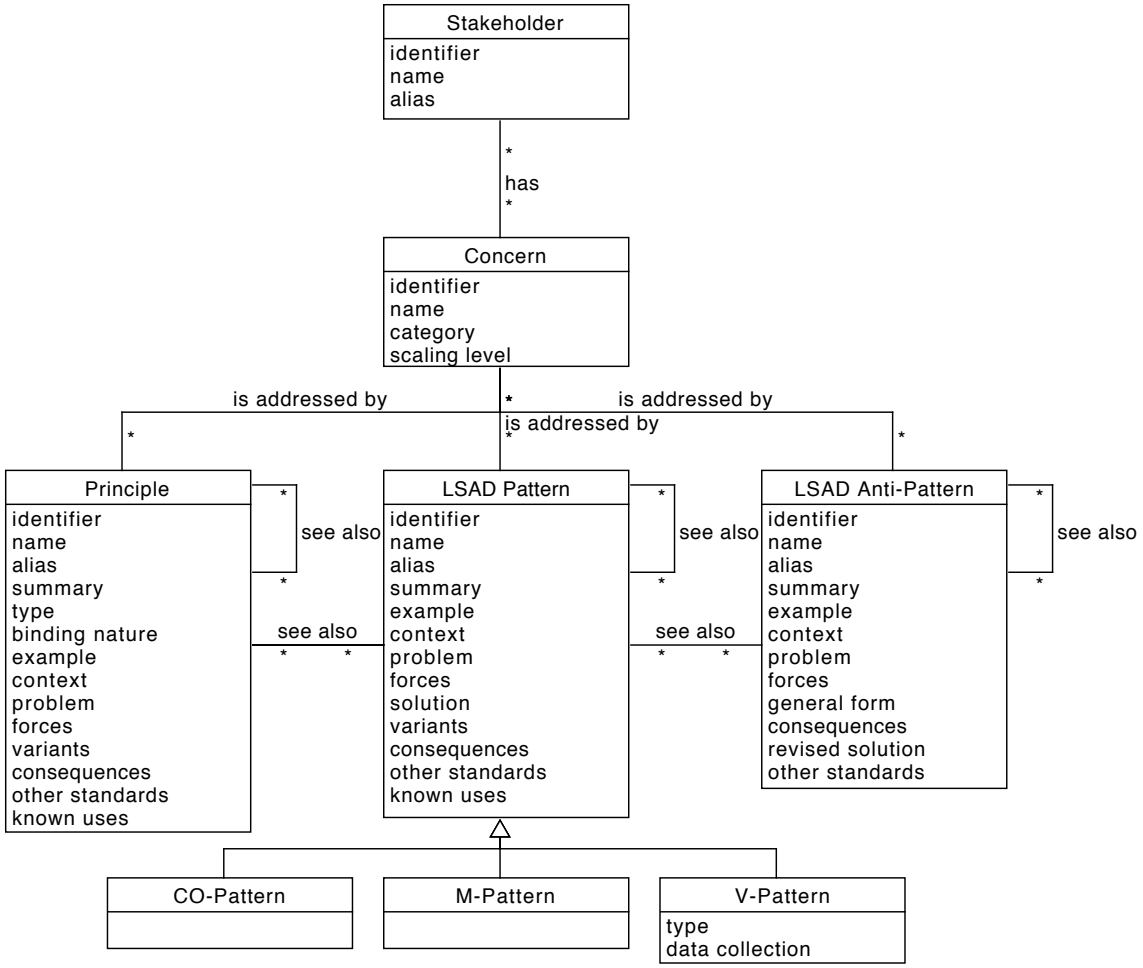


Figure 2.2.: The Large-Scale Agile Development Pattern Language by [74]

The formal definition of the LSADPL is depicted in Figure 2.2. The key concepts are explained in the following, based on the descriptions by the authors of the LSADPL [74].
Concern. Concerns are tasks or challenges of certain stakeholders. A Concern is categorized either as *communication & coordination* or *methodology* in this thesis. However, the LSADPL as described in [74] offers various additional categories. A Concern can be addressed by one or multiple Patterns, Anti-Patterns, and Principles.
Stakeholder. A Stakeholder can be any person that has some kind of interest or involvement in the project. Stakeholders may have Concerns in their work.
Pattern. A Pattern is a pattern as formally defined in Section 2.1. In the LSADPL, a Pattern can be categorized as *methodology* (M-Pattern), *coordination* (CO-Pattern), or *viewpoint* (V-Pattern), depending on which kind of solution it offers [74]. Methodology Patterns describe concrete steps to address a concern. Coordination Patterns describe coordination mechanisms (see also Section 2.4). Viewpoint Patterns describe visualizations.
Anti-Pattern. An Anti-Pattern is a Pattern that may sound promising to solve a concern, but it actually does not solve the problem. An Anti-Pattern describes common mistakes in the solution section. Additionally, it provides a revised solution section which describes how to avoid these mistakes to actually solve the problem [74].
Principle. Principles provide steps or guidelines to address a certain Concern [74]. However, they do not necessarily provide a viable solution to the Concern, but rather an assistance on the way to solve a Concern.

Because the LSADPL Pattern is a key concept for this thesis, the different sections of a Pattern are now described in detail according to the documentation template from [74].
Identifier, Name, Alias. Each Pattern starts with an unique identifier. Further, the Pattern has an unique, descriptive name as well as an alias section for other known names of this Pattern.
Summary. A short summary of the solution provided by the Pattern.
Example. A story that showcases a situation where the Pattern is used.
Context. Description of the situation in which the addressed Concern rises and the Pattern is applicable.
Problem. References one or more Concerns that are addressed by the solution of this Pattern.
Forces. A description of the forces that drive the problem and make it hard to solve.
Solution. The heart of the Pattern is the solution that solves the referenced Concerns in the given context. The solution is described in this section.
Consequences. Every Pattern implies benefits as well as liabilities when being applied. They are listed in this section.
Variants, Known Uses, See Also, Other Standards. These sections provide references to various other sources. 'Variants' describes possible variations of the Pattern. 'Known Uses' lists real cases where the Pattern is applied. 'See Also' holds references to other LSADPL patterns. 'Other Standards' points to other sources or frameworks where concepts similar to the Pattern are documented.

2.3.4. The Nexus Framework

As explained in Section 2.2.3, Scrum focuses on small, co-located teams. But in practice often more than one team are working together on complex products. Throughout the last few years, due to the proven benefits of agile methodologies, large companies have gained increasing interest in applying agile methodologies at their bigger projects as well [26]. Thus, scaling agile frameworks have emerged that guide users to apply agile practices in larger projects and organizations. While all observed teams of this study did use Scrum,

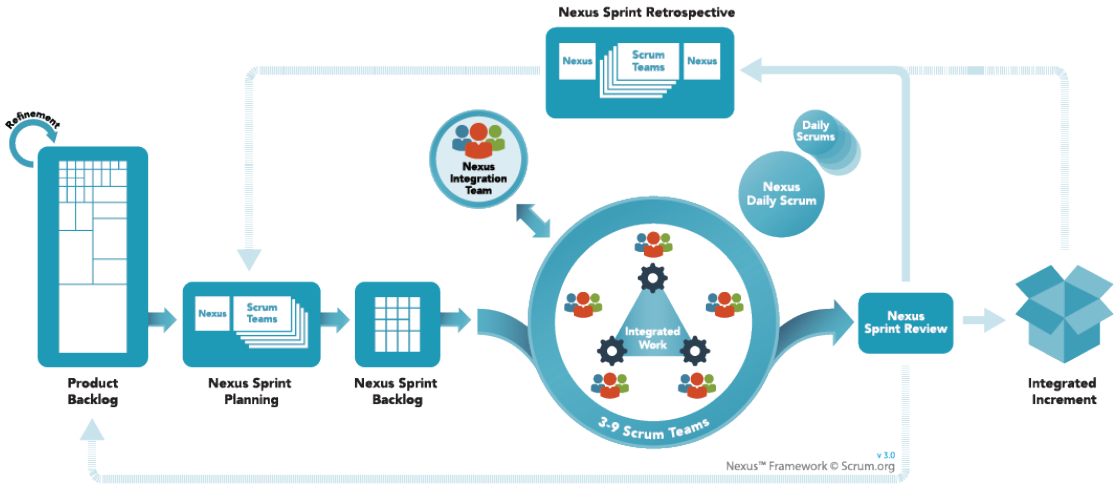


Figure 2.3.: Nexus Framework illustration by [66]

the case organization did not use any scaling agile framework. However, towards the end of this study in August 2019, the teams of Product A (see Section 4.2) started piloting using the Nexus framework for their inter-team coordination. They decided to do so because the overall project had grown significantly. Therefore, in this section the Nexus framework is described.

The Nexus framework is intended to apply Scrum in a scaled environment [11]. The Nexus framework is focused on multiple Scrum teams working together on a single product backlog, team coordination, and dependencies between teams [11]. Just like Scrum, Nexus comes with roles, events, and artifacts [66]. They are explained in the following. The Scrum roles, events, and artifacts, which still apply in the Nexus framework, are not mentioned again. They can be looked up in the Scrum description in Section 2.2.3.

Nexus Sprint Backlog. In Nexus, each team continues to have its own Scrum sprint backlog. To make a plan for the sprint and to make transparent what is done by which team during the sprint, Nexus adds the Nexus Sprint Backlog artifact [66]. It consists of the individual team sprint backlogs [66].

Nexus Integration Team. Nexus adds the role of the Nexus Integration Team [66]. The

Nexus Integration Team is responsible for coordination and integration in the Nexus, but also for supervision of the application of Scrum and Nexus [11]. It usually consists of a Scrum Master, the Product Owner, and other members [66].
Nexus Sprint Planning. In the Nexus Sprint Planning, representatives of each Scrum team select tasks for the upcoming Sprint for the teams [66]. Each team then plans their work in individual Scrum sprint plannings [66]. The overall sprint goals and work assignments are visualized using the Nexus Sprint Backlog [66].
Nexus Daily Scrum. Just like in Scrum, there is a short daily meeting. In the Nexus Daily Scrum, team representatives discuss integration progress and possible dependencies between the work of their teams [11]. If any problems are encountered, the individual teams can further discuss how to address them in their own daily scrum meetings [66].
Nexus Sprint Review. At the end of each sprint, the new product increment is reviewed and feedback is gathered in the Nexus Sprint Review [66]. All teams and stakeholders take part in the meeting and the product backlog may be modified if necessary [66]. Because individual teams might not be able to create a viable product increment on their own, the Scrum Sprint Review for the Scrum teams is removed and replaced by the Nexus Sprint Review [11].
Nexus Sprint Retrospective. The Nexus Sprint Retrospective includes three steps [66]. In the first step, representatives of the teams meet to discuss common challenges of the last sprint. Second, each team holds their individual Scrum sprint retrospective to discuss these challenges. Finally in the third step, the representatives of the teams meet again to discuss measures that have been proposed by the teams.

Other Scaling Agile Frameworks

Additionally to the Nexus framework, there are several other scaling agile frameworks. Some interviewees in the study mentioned they had made experiences with Scaled Agile Framework (SAFe®) in the past. SAFe® is the most widely used scaling agile framework, followed by Scrum of Scrums [40]. A comparison of popular scaling agile frameworks, including SAFe®, Scrum of Scrums, and others, has been created by Ebert and Paasivaara [30]. A more extensive evaluation and comparison of 20 scaling agile frameworks has been conducted by Karabacak [42].

2.4. Coordination Theory

Because this thesis focuses, among others, on coordination concerns and practices, the following section explains the theoretical background of work coordination in organizations.

Coordination is the management of dependencies [50, 63]. Accordingly, coordination can be achieved by characterizing and grouping dependencies into different categories, and applying mechanisms to manage these dependencies [50]. Examples for basic dependency categories where coordination processes are needed are tasks depending on limited

shared resources, sequential consumer-producer workflows, or task-subtask relationships [50]. Besides the management of dependencies, Malone and Crowston [50] also describe the *group decision making* and *communication* processes as being important for coordination. These two coordination processes can also be seen as management of dependencies. However, [50] explicitly explain them because of their importance. Figure 2.4 shows the coordination processes by Malone and Crowston [50]. They state that all the mentioned instances of coordination comprise of *actors* performing *activities* that are *interdependent* [50].

Coordination Process	*Dependency Management*	*Group Decision Making*	*Communication*
Examples	• Managing limited shared resources • Managing sequential producer / consumer relationships • Managing parallel workflows • Managing task / subtask relationships	• Deciding how to allocate resources • Deciding how to segment tasks	• Developing standards for communication • Establishing common knowledge

Figure 2.4.: The coordination processes by Malone and Crowston [50]

Van De Ven et al. [77] define a categorization of coordination mechanisms. They present three alternative modes that are used by organizations to coordinate their work activities. These modes are *impersonal mode*, *personal mode*, and *group mode* [77]. All plan-driven coordination such as schedules, rules, and policies are categorized in the *impersonal coordination mode*. They are codified and the required actions are coordinated impersonal by these codes [77]. In contrast to the plan-driven *impersonal coordination mode*, the other two coordination modes, *personal mode* and *group mode*, are based on mutual adjustment [77]. This means they factor in feedback [73, 77]. The *personal* coordination is done by individual persons or roles via vertical or horizontal communication channels. Individuals adjust their work based on personal interaction with others. The *group mode* relies on scheduled and unscheduled coordination mechanisms for groups, such as communication events and group meetings [77]. Further, Van De Ven et al. [77] also identify three determinants that affect which coordination mode — or combination of coordination modes — is used in specific situations. These determinants are *task uncertainty*, *task interdependence*, and *size of work unit* [77]. The more difficult and variable a task is, the more difficult it is to coordinate. Therefore, personal mode and group mode are of increasing importance. Similarly, higher interdependence between different work flows is accompanied by increases in group mode coordination. Finally, increased size of work units leads to increased usage of the impersonal coordination mode [77].

Espinosa et al. [32] categorize the three coordination modes by Van De Ven et al. [77] as *explicit* coordination. They add the category of *implicit* coordination [32]. Implicit coordination mechanisms are existing in the shared team cognition, but are not applied purposely for coordination of activities [32]. Such implicit mechanisms can be assumptions about the other teams' tasks or knowledge about the task and the team [32]. Figure 2.5 shows a combination of the three coordination modes by Van De Ven et al. [77] and the additional classification by Espinosa et al. [32].

Coordination Mode	*Explicit*			*Implicit*
	Impersonal Mode	*Personal Mode*	*Group Mode*	
Examples	• Plans • Schedules • Policies • Information & Communication Systems	• Individual role occupants	• Staff meetings or committees • Scheduled or unscheduled	• Common assumptions • Shared knowledge • Prior experience
Effect of Determinant Increase on Usage	↓ Task Uncertainty ↑ Task Interdependence ↑ Size of Work Unit	↑ Task Uncertainty ↑ Task Interdependence ↑ Size of Work Unit	↑ Task Uncertainty ↑ Task Interdependence ↓ Size of Work Unit	
Classification	***Programming***	***Mutual Feedback***		

Figure 2.5.: The three coordination modes by [77] (blue), combined with the classification by [32] (white)

3. Related Work

This chapter summarizes existing literature on topics related to the objective of this thesis. First, Section 3.1 presents related literature on the topic of scaling agile development practices. Then, Section 3.2 summarizes related work on coordination of large-scale software development work. Section 3.3 compares other pattern languages to the LSADPL, which is used in this thesis for structuring the findings.

3.1. Related Work on Large-Scale Agile Development

As already described in the introduction, the topics "agile and large projects" and "inter-team coordination" are of interest for practitioners [25, 33]. Following this research interests, Laanti published a paper in 2014 examining the characteristics and principles of scaled agility [46]. In a first step, Laanti [46] identifies eight aspects of agility that should be considered when scaling agile in an organization. The Strategic and Business Agility, as well as the Agile Organization aspect, focus on combining the business strategy and organizational structures with agility [46]. The People Agility and Organizational Culture are aspects regarding the adoption of agile values by the people. They aim at aligning the organizational values with the agile values [46]. The Tools Agility aspect focuses on having a work environment and tool-set that enables agile working in the organization [46]. The Product Agility relates to being agile in what to build [46]. The Payoff Agility aspect refers to being agile in where to invest the available money [46]. In the second step of the paper, Laanti [46] presents a list of 21 principles for scaled agility. For each of the principles an explanation is provided and the values it originates from are explained.

Dikert et al. [23] conducted a systematic literature review focusing on challenges and success factors of adoptions of agile methods at scale. For their literature review, Dikert et al. [23] defined large-scale software development as including at least 50 people or at least 6 teams. The result of the literature review is a list of 35 challenges divided into nine challenge categories, and 29 success factors divided into eleven success factor categories. The challenge categories are *Change resistance*, *Lack of investment*, *Agile difficult to implement*, *Coordination challenges in multi-team environment*, *Different approaches emerge in a multi-team environment*, *Hierarchical management and organizational boundaries*, *Requirements engineering challenges*, *Quality assurance challenges*, and *Integrating non-development functions*. The success factor categories identified by Dikert et al. [23] are *Management Support*, *Commitment to change*, *Leadership*, *Choosing and customizing the agile approach*, *Piloting*, *Training and coaching*, *Engaging people*, *Communication and transparency*, *Mindset and Alignment*, *Team auton-*

Focus Area	Identified Characteristics
Customer Involvement	(1) Solution description was being developed using teamwork (2) Continuous and iterative customer involvement (3) Boundaries between need analysis, solution description and development were blurred
Software Architecture	(1) Tension between up-front and emergent architecture (2) Demanding architect role in large-scale projects
Inter-Team Coordination	(1) Coordination was achieved by combination of arenas (2) Use of coordination arenas changed over time (3) Importance of informal coordination arenas

Table 3.1.: The key characteristics of the case program in the three focus areas studied by Dingsøyr et al. [27]

omy, and *Requirements management*. In their literature review Dikert et al. [23] also found that Scrum was the most frequently used agile methodology in the reviewed literature, followed by XP. Combinations of those two methodologies as well as with other methods were quite commonly used [23].
The work by Dikert et al. [23] on challenges and success factors for scaling agile development was not used as a foundation for this thesis because it already served as an input for the literature review of Uludağ et al. [75]. This literature review [75] in turn did serve as a foundation for this thesis.

Dingsøyr et al. [27] conducted a case study at a large agile software development program. They investigated how agile methodologies have been adapted to fit the large-scale setting and to complement with non-agile practices. The studied case consisted of 12 co-located teams, with 175 people being involved. The project had been finished by the time the research was conducted. The focus areas of their case study were customer involvement, software architecture, and inter-team coordination in scaled agile development. Three organizations did take part in the case project. Thus, Dingsøyr et al. [27] conducted a group interview for each focus area with people from each participating organization respectively. As a second type of data they studied documents. For each of the studied focus areas of the case organization, Dingsøyr et al. [27] identified several key characteristics. These key characteristics are listed in Table 3.1.

3.2. Related Work on Coordination

In Section 2.4 the theoretical basics of coordination were discussed. This section focuses on related work that has been created on coordination in large-scale agile software development. In agile software development intra-team coordination is achieved by team-level mechanisms, such as the Daily Scrum meetings or Sprint Plannings. This section focuses

on related work about large-scale development environments and inter-team coordination.

Dingsøyr et al. [28] conducted a study at two large software development programs. Their study focused on how group mode in the form of unscheduled and scheduled meetings is applied in inter-team coordination and how it changes over time. They state, that the mutual adjustment coordination (personal and group mode) in large software projects usually comes with hierarchy and bureaucratic structures [28]. The resulting group mode coordination is therefore often implemented using layered adjustment structures, such as Scrum of Scrums meetings 'on top' of team-internal Scrum meetings [28]. They conclude that group mode coordination is essential in large-scale agile software development projects. They especially highlight the importance of unscheduled meetings for inter-team coordination and knowledge exchange [28].

Nyrud and Stray [55] conducted a case study to investigate which coordination mechanisms are used for inter-team coordination in large-scale agile software development. For their study, they selected a digitization project. They identified eleven different coordination mechanisms and mapped them onto the three coordination modes [55]. The observed project used five impersonal, one personal, and six group mode coordination mechanisms. The mechanism *Informal Ad Hoc Communication* is categorized as personal mode and group mode, hence the sum of eleven identified mechanisms. Figure 3.1 depicts their mapping of the mechanisms onto the coordination modes. We present a similar mapping at the findings of this thesis in Section 4.5.

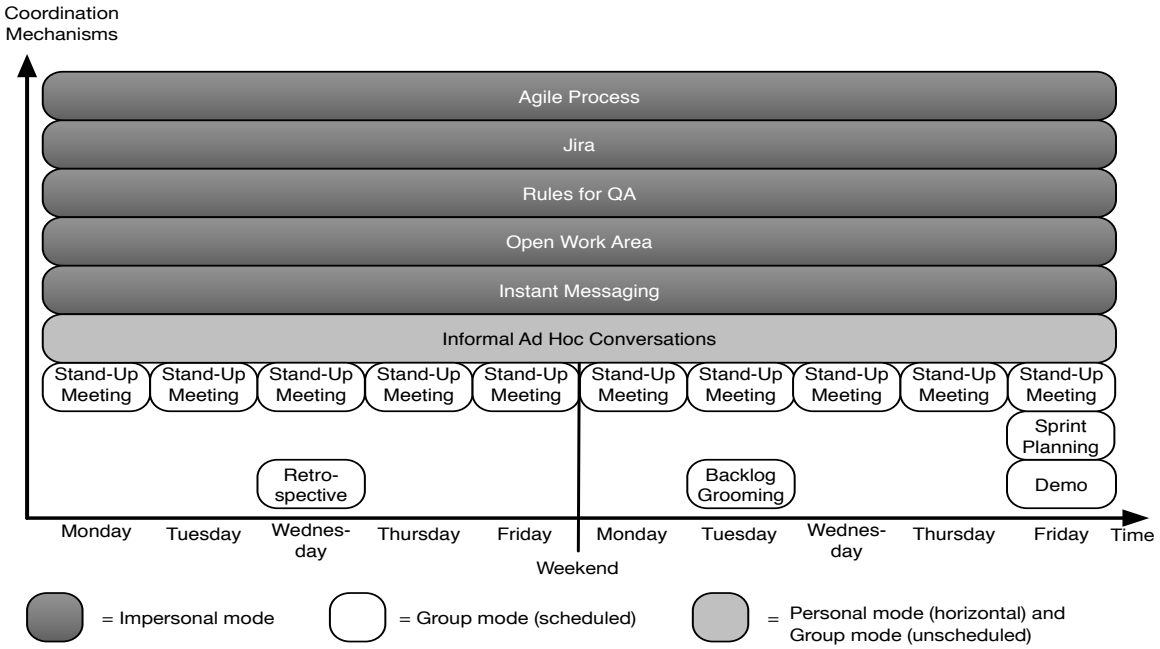


Figure 3.1.: Mapping of the identified coordination mechanisms by [55]

Bick et al. conducted a multiple case study at SAP SE, to research how inter-team coordination is achieved in large-scale agile software development [9]. They conducted 68 interviews and identified five different coordination mechanisms that can be used to scale agile work [9]:

- Coordination via a central, dedicated organizational unit.
- Proxy collaboration via meetings of team representatives at the next higher hierarchy level.
- Centralized plannings that use decentralized input from all development teams.
- Full collaboration by running a planning workshop with all members of the development program.
- Need based ad hoc communication.

They conclude that proactive dependency management between development teams is beneficiary and possible, both in settings of top-down and bottom-up coordination [9].

Paasivaara et al. [56] conducted a multiple case study with two organizations. They interviewed 58 practitioners about how their organizations handled the inter-team coordination in the large-scale distributed Scrum setups. In the study, they found that both organizations were relying on a layered structure of Scrum-of-Scrums (SoS) meetings. The first organization had localized Scrum-of-Scrums meetings followed by a global SoS meeting. The second organization used feature specific SoS in combination with a global SoS meeting. Paasivaara et al. [56] conclude, that in both cases the global SoS meetings worked poorly because of disjoint work areas and interests of the participants. In contrast, they find that the localized and feature specific SoS meetings were being perceived as generally useful [56].

3.3. Related Work on Pattern Languages

Similar to the LSADPL, other research projects have worked on formalizing practices in pattern languages. This section gives an overview of pattern languages for agile software development, large-scale software development, or both areas. The initial authors of the LSADPL have also created a similar overview [74]. They focus on the works of Coplien [19, 21], Harrison [37], Taylor [71], Elssamadisy [31], Välimäki [76], Beedle [6, 7], and Mitchell [53]. The overview following here is extending this comparison with additional sources of pattern languages.

Ambler. Process Patterns: Building Large-Scale Systems Using Object Technology. 1998.
In this work [3], Ambler presents a pattern language containing process patterns to apply object-oriented software development in large-scale environments. The patterns are targeted at medium to large organizations [3]. Ambler defines his process patterns as activities for developing object-oriented software, that describe practices to follow but are not detailed descriptions of how to implement them. The work does not use or reference

any agile approaches. Instead, a custom software development process is created, that is following a linear structure in general. This process consists of the four phases *Initiate*, *Construct*, *Deliver*, and *Maintain*. Each phase itself is structured iteratively, resulting in a serial process that consists of iterative phases. According to this hierarchy, the patterns are structured into phase patterns, stage patterns, and task patterns. In this work [3], Ambler describes the patterns for the first two phases of the object-oriented software process, *Initiate* and *Construct*.

Ambler. More Process Patterns: Delivering Large-Scale Systems Using Object Technology. 1999.
This work by Ambler [2] is the extension of [3]. It describes the second part of the patterns in Ambler's object-oriented software process for the phases *Deliver* and *Maintain*. The focus, categorization, and structure of the pattern language and its patterns is the same as in [3].

Bozheva, Gallo. Framework of Agile Patterns. 2005.
Bozheva and Gallo [14] present a framework for patterns with specific consideration of patterns that originate from agile methodologies. However, the pattern language is not focused on large-scale development settings. The framework not only consists of 39 patterns, but also contains seven *Principles*, eleven *Concepts*, and relationships in the form of *invokes* and *supports*. In the framework, the activities of one pattern can invoke other patterns, and the implementation of one pattern can be supported by other patterns [14]. Concepts are similar to patterns and share their attributes, except that there are no activities associated with them. They are used to describe essential characteristics of important concepts in agile methodologies. Principles are people-oriented and flexible rules. They are described with the attributes *Intent*, *Origin*, and *Guidelines*. Compared to the LSADPL, this framework shows some similarities. Both languages have patterns, principles, as well as relationships between them [14, 74]. However, while the LSADPL holds an explicit class for stakeholders [74], the framework by Bozheva and Gallo [14] attributes roles to patterns.

Lescher. Patterns for Global Development: How to Build One Global Team? 2010.
The patterns created by Lescher [49] focus on building a distributed, global team and facilitating efficient communication and collaboration among the team members and locations. The goal of the patterns is to deal with the problems of geographic distribution such as local sub-grouping and missing trust between team members [49]. The five patterns all stem from the global development efforts made at Siemens, a large German multi-industry company. Lescher [49] groups the practices into two categories, one focusing on team-building and the other focusing on communication. The work focuses on large-scale software development but is not specifically for agile setups.

Monasor et al. Towards a Global Software Development Community Web: Identifying Patterns and Scenarios. 2013.
In contrast to the other presented related pattern languages, Monasor et al. [54] do not identify any patterns in their work directly. Instead, they developed a method for collecting practice oriented scenarios and patterns regarding Global Software Development. This should make them more applicable for practitioners than generalized patterns identified by other research [54]. The work is presented as a Community Web and focuses on *Pedagogical Patterns* that help practitioners to build their own set of practices on top of them [54]. Anti-Patterns can also be part of the collected scenarios and patterns. Categories are not part of the work, instead a hierarchical tree-structure is used to arrange the patterns. This hierarchy is extensible horizontally and vertically, i.e., patterns can be added and grouped using the tree-structure. Additions and extension can be done by contributors [54]. The method does not focus on any specific development process model, but is applicable to any used development process [54]. Thus, agile methodologies are not a region of focus.

Sutherland et al. Teams that Finish Early Accelerate Faster: A Pattern Language for High Performing Scrum Teams. 2014.
Sutherland et al. [70] state that Scrum can be explained in its basics in only two minutes. They show nine patterns that reflect the very core of the Scrum practices [70]. The goal of the work is to increase the amount of successful projects which are using the Scrum methodology [70]. The patterns address the major problems that teams are facing during the application of Scrum in the development process. They are arranged into three categories: patterns to get ready for the sprint, patterns to deal with common disruptive problems during a sprint, and patterns to achieve the *hyper-productive* state [70]. The patterns are not formatted in a strict way, but are rather presented as mostly free-text. The clear focus of the work is on the agile Scrum methodology, but no focus is on large-scale situations.

Kausar, Al-Yasiri. Distributed Agile Patterns for Offshore Software Development. 2015.
Kausar and Al-Yasiri [44] have conducted a literature review to identify distributed agile patterns. They focus on development projects making use of offshoring to other countries and agile methodologies. In contrast to *normal* large-scale development, distributed offshore development often makes use of benefits such as lower wages, higher availability of qualified work personnel, and market proximity in the offshoring countries [44]. However, there are also many similarities between large-scale development and offshoring scenarios. In both cases the development teams have to deal with additional concerns such as trust, communication and coordination, time-zones, and cultural issues [44]. Further, Kasuar and Al-Yasiri lay a strong emphasize on using agile methodologies in the considered scaled development environment [44]. In their literature review they identified 15 *distributed agile patterns*. The patterns were reviewed and verified in interviews with companies [44]. The patterns are arranged into four categories: *Management patterns*, *Communi-*

cation patterns, *Collaboration patterns*, and *Verification patterns* [43]. The catalog of practices focuses on addressing the concerns of distributed offshore software development while using agile methodologies [44].

The following tables show all the explained pattern languages (Tables 3.2 and 3.3) and also the pattern languages compared by the authors of the LSADPL [74] (Tables 3.4, 3.5, and 3.6). For each pattern language, the pattern attributes are shown. The content, scope, and categories of the pattern languages are also shown again in the tables.

Table 3.2.: Comparison of related pattern languages

	Ambler [2, 3]	**Bozheva, Gallo [14]**	**Lescher [49]**
Pattern Attributes	Name, Free text description, Initial Context: Entry Conditions, Solution(s), Resulting Context: Exit Conditions, Secret of Success, Process Checklist, What You Have Learned in This Chapter, Refernces and Recommended Reading	Name, Origin, Category, Application Scenario, Roles, Activities, Tools, Guidelines	Name, Motivation, *** Context, Problem, Forces, Therefore: Solution, *** Resulting context, Related patterns
Scope & Goal	Patterns for medium to large-scale object-oriented software development	Patterns originating from different agile methodologies with special focus is on rationale for applying the patterns	Patterns to build a globally distributed team and make communication and collaboration effective
Focus on agile	No	Yes	No
Focus on large-scale	Yes	No	Yes
Number of Patterns	18	39	5
Categories	(1) Task process patterns (2) Stage process patterns (3) Phase process patterns	(1) Implementation & Testing (2) Design (3) Resource Organization (4) Contract Management (5) Software Process Improvement (6) Project and Requirements Management	(1) Kick-Off and Teambuilding (2) Project Communication

	Monasor et al. [54]	**Sutherland et al. [70]**	**Kausar, Al-Yasiri [44]**
Pattern Attributes	Name, Problem, Population, Analysis of the Problem, Solution, Reference/Source	Name, Summary, free text section	Name, Intent, Also Known As, Category, Motivation, Applicability, Participants, Collaboration, Consequences, Known Uses, Related Patterns
Scope & Goal	Pattern language for pedagogical patterns in Global Software Development, that can help to build a custom set of practices	Patterns for problems of Scrum teams that hinder them to finish early	Patterns for distributed teams using agile methods with special focus on offshoring scenarios
Focus on agile	No	Yes	Yes
Focus on large-scale	Yes	No	Yes
Number of Patterns	-	9	15
Categories	Hierarchical structure with extensible number of categories	(1) Patterns to get ready for a sprint (2) Patterns to deal with disruptive problems during a sprint (3) Patterns to become hyper-productive	(1) Management patterns (2) Communication patterns (3) Collaboration patterns (4) Verification patterns

Table 3.3.: Comparison of related pattern languages

	Harrison [37]	**Beedle et al. [7]**	**Taylor [71]**
Pattern Attributes	Name, Problem, Forces, Solution, Rationale, Variation, Related Patterns	Name, Context, Misfit variables, Problem, Forces, Solution, Examples, Resulting Context, From, Rationale	Name, Intent, Context, Problem, Forces, Solution, Resulting Context, Known Problems, Related Patterns
Scope & Goal	Collection of patterns for creating effective software development teams	Small collection of Scrum patterns	Collection of patterns for creating product software development environments
Focus on agile	No	Yes	No
Focus on large-scale	No	No	No
Number of Patterns	4	3	9
Categories	-	-	(1) Establishing a Production Potential (2) Maintaining a Production Potential (3) Preserving a Production Potential

Table 3.4.: Comparison of related pattern languages based on [74]

	Coplien, Harrison [21]	**Elssamadisy [31]**	**Beedle et al.[6]**
Pattern Attributes	Name, Image, Short story, Context, *** Problem summary, Forces, *Therefore* Solution, *** Disucssion, Related Patterns, Confidence level	Name, Description, Dependency Diagram, Business Value, Sketch, Context, Forces, Therefore, Adoption, But, Variations, References	Name, *** Context, *** Problem, Forces, Solution, Resulting context, Related Patterns
Scope & Goal	Collection of organizational patterns that are combined into a collection of four pattern languages	Collection of patterns for successfully adopting agile practices	Collection of the most essential best practices of Scrum
Focus on agile	Yes	Yes	Yes
Focus on large-scale	No (but authors mention some patterns are applicable in large-scale)	No	No
Number of Patterns	94	38	11
Categories	(1) Project Management (2) Piecemeal Growth (3) Organizational Style (4) People and Code	(1) Feedback Practices (2) Technical Practices (3) Supporting Practices (4) The Clusters	-

Table 3.5.: Comparison of related pattern languages based on [74]

	Välimäki [76]	**Mitchell [53]**	**ScrumPLoP [69]**
Pattern Attributes	Identifier, Name, Initial context, Roles, Forces, Solution, Resulting context	Name, Intent, Proverbs, Also Known As, UML Diagram, Applicability, Motivation, Structure, Implementation, Consequences	Irregular
Scope & Goal	Enhancing performance of project management work through improved global software project management practice	Patterns for agile transformations and initial implementation of agile practices	Body of pattern literature around agile and Scrum communities
Focus on agile	Partially	Yes	Yes
Focus on large-scale	Yes	No	No
Number of Patterns	18	49	234
Categories	(1) Directing a Project (2) Starting up a Project (3) Initiating a Project (4) Controlling a Stage (5) Managing Stage Boundaries (6) Closing a Project (7) Managing Product Delivery (8) Planning	(1) Patterns of Method (2) Patterns of Responsibility (3) Patterns of Representation (4) Antipatterns	(1) Value Stream (2) Team (3) Sprint (4) Process Improvement (5) Product Organization (6) Distributed Scrum (7) Scaling Scrum (8) Scrum Core (9) Misc

Table 3.6.: Comparison of related pattern languages based on [74]

4. Identifying Recurring Concerns and Practices

In the preceding chapters the foundations of this thesis and related work were presented. The LSADPL, which forms the basis of this work, was explained. The existing work on concerns in large-scale agile development gives a starting point for identifying further concerns. This chapter describes the study that is used to identify concerns and pattern candidates for the scaling of agile methodologies. Sections 4.1 and 4.2 first lay out the methodology that is used to collect the data and describe the organizational setup of the industry partner. Section 4.3 lists all concerns that are identified in the study and identifies recurring concerns. Finally, Sections 4.4 and 4.5 present the identified pattern candidates (good and bad practices) and map them into a coordination mode categorization afterwards.
Throughout the following sections, concerns and pattern candidates are addressed using unique identifiers (IDs). Concern identifiers start with a capital '*C*'. Coordination practice identifiers start with '*CO*'. Methodology practice identifiers start with '*M*'. Viewpoint practice identifiers start with '*V*'. And anti-pattern candidates start with '*AP*'.

4.1. Methodology

The data collection to identify recurring concerns and practices incorporates multiple techniques of data gathering [60]. Third degree data was collected using internal documentation and artifacts of the industry partner organization. This includes the corporate wiki, Jira and ZenHub boards, workshop or presentation slides, and others. Second degree data was collected by observing the workflows and meetings of teams within the studied organizational units. First degree data is the most important part of the data collection for this study. It was collected by conducting semi-structured interviews with 15 practitioners from the case organization [60, 80].

The semi-structured interviews followed a frame that is illustrated in Figure 4.1. Each interview started with predefined questions about the interviewee's personal background, experience with large-scale agile development, and the current role in the team. Then, we asked the interviewees to identify their three most pressing concerns which they are facing in their current role, regarding the methodology and coordination of work in the large-scale agile development setting. For every concern that was mentioned, we asked the interviewees to describe a solution that they applied in practice to address this concern. In the third step of the interviews, we showed a list of concerns to the participants that we

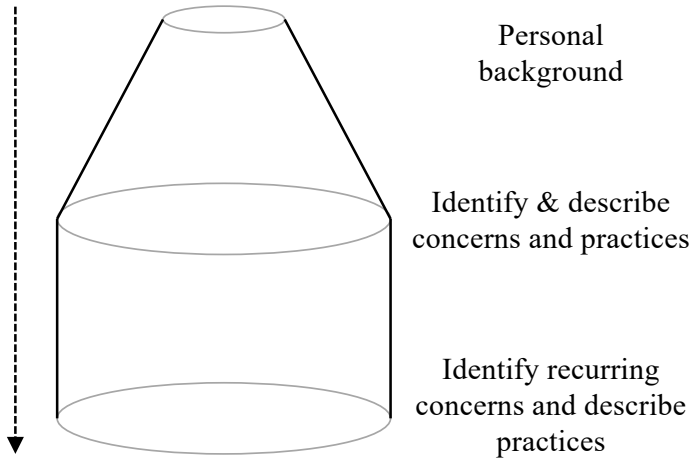


Figure 4.1.: Visualization of the used interview structure, an adaption of the pyramid principle by [60]

already had identified previously. We asked them to select those concerns that they are also facing and choose any concerns to again describe a solution they applied in practice.

Each interview lasted approximately one hour, was audio-recorded, and was transcribed afterwards. To reduce the potential for bias introduced by the transcribing researcher, we handed out the transcript and artifacts that we generated from the interviews to the respective participants for a review. The roles and years of practical experience with scaled agile development of the interviewees are listed in Table 4.1. Each interviewee is assigned an alias which is used in the text to refer to the specific table entry.

We identified six teams from the Munich based product areas of the case organization as the major sub groups that are available and relevant to this study. They are described in detail in Section 4.2. We sampled interviewees out of these six teams. Both, the sampling and the list of concerns, which we presented to participants in the third part of the interviews, were subject to the *rolling quality* of a single-case, embedded study [52]. This means that we selected the interviewees on a rolling basis during the study and presented them a list of concerns that also contained ones mentioned by previous interviewees. To collect an equal amount of opinions on all identified concerns, during the interview recap session we asked participants to identify concerns that are relevant to them from a list of concerns that had been collected after their personal interview took place.

We imported the collected data of all three levels into a qualitative data analysis software (MAXQDA 2018) to conduct the analysis. The data was coded and analyzed following the five steps described by Cruzes and Dybå [22]. Significant segments of the transcripts, protocols, and documentation text were labeled with concept codes in the qualitative data analysis software [52]. We assigned codes following the integrated approach [22]. This means, the existing list of concerns from literature served as a start list of provisional codes. Also, the LSADPL acted as a general accounting scheme [22, 52]. Thus, we categorized the

No.	Alias	Role	Experience	Team	Product Area
1	AC1	Agile Consultant	6 - 10 years	-	-
2	D1	Developer	1 - 2 years	Team4	B
3	D2	Developer	1 - 2 years	Team6	C
4	D3	Developer	6 - 10 years	Team3	A
5	M1	Development Manager	3 - 5 years	- / Team5	B
6	M2	Development Manager	16 - 20 years	-	A
7	PO1	Product Owner	1 - 2 years	Team4	B
8	PO2	Product Owner	1 - 2 years	Team2	A
9	PO3	Product Owner	11 - 15 years	Team1	A
10	SM1	Scrum Master	3 - 5 years	Team2	A
11	SM2	Scrum Master	3 - 5 years	Team5	B
12	SM3	Scrum Master	1 - 2 years	Team1	A
13	SA1	Software Architect	3 - 5 years	Team1	A
14	SA2	Software Architect	6 - 10 years	Team2	A
15	D4	Developer (Tech Lead)	11 - 15 years	Team6	C

Table 4.1.: Overview of the interview participants

codes according to the entities of the LSADPL. In the next step, we consolidated codes and combined them in cases of overlaps. Finally, we created the artifacts presented in this thesis based on the coding results. To ensure the objectivity and quality of the data analysis, the coding was reviewed by a second researcher. Table 4.2 displays an overview of the used code structure, assigned codes, and identified artifacts.

Code Category	# Identified Elements	# Codes
Concerns	39	921
Methodology Pattern Candidates	9	142
Coordination Pattern Candidates	23	123
Viewpoint Pattern Candidates	6	53
Anti-Pattern candidates	4	42
Principle candidates	2	18
	Total	1299

Table 4.2.: Overview of the used code structure and number of identified elements

4.2. Case Description

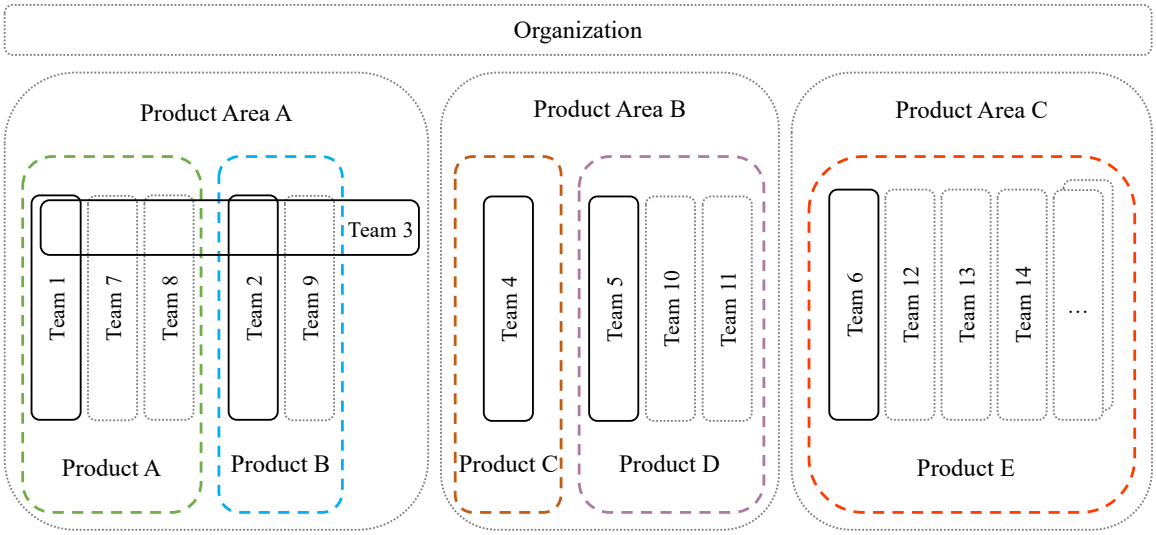


Figure 4.2.: The team setup in the case organization as of July 2019

We collected the data in a productive software development environment at a large German software vendor. The unit of analysis was one development site, which housed multiple product areas. We observed three of those product areas during the course of this study. Therefore, the study can be categorized as single-case embedded [80]. The three product areas are assigned the aliases "A", "B", and "C". They are segmented based on the products they focus on. Each product area has a Development Manager and multiple teams. Figure 4.2 shows the team constellation as of July 2019. The observed product areas of the company did not use any scaling agile framework such as SAFe® or Nexus during the study. However, interviewees AC1 and SA2 mentioned that there have been attempts to use SAFe® in other parts of the organization. The observed teams are all structured according to the Scrum setup [67], except Team3.

Product Area A: Team1
Team1 has 14 team members, including a full-time Scrum Master and PO, the development team of 10 people, an affiliated Software Architect, and one dedicated DevOps engineer from Team3. The Team1 was set up completely from scratch in August 2018. Throughout 2018, the team grew constantly. Initially, Team1 was developing a solution to make business objects, which are maintained by several products of the organization, reusable across the whole product suite. However, the project vision and requirements were not clear when the team started in August 2018. Therefore, Team1 faced several changes in project scope. This is clearly reflected in the identified concerns in Section 4.3. In March 2019, the project of Team1 was restructured completely and was merged into a new, sig-

nificantly larger project to develop Product A. Since March 2019, Team1 is collaborating with Team7, which is located in Walldorf, and Team8, which is located in Potsdam. The new project is not restricted to business objects, but also encompasses several other kinds of resources. As the teams working on Product A were constantly growing throughout the course of this study, they decided to implement the Nexus framework starting from end of July 2019. Due to the limited time scope of this thesis the implementation of the Nexus framework could not be considered anymore.

Product Area A: Team2
Team2 has seven team members, including a part-time Scrum Master that is also doing development, a full-time PO, a development team of three people (including part-time Scrum Master), a Technical Writer, an affiliated Software Architect, and a DevOps Engineer from Team3. Team2 is developing a cloud service for authentication and authorization (Product B) that will be used within the whole product suite. The Product B is intended to replace the product-level identity management with suite-level identity management. Team2 are collaborating with people from Team9 in Walldorf. Further, during the course of this study Team2 was assisting in development of another project for a period of three months. The other project was facing capacity problems towards an approaching deadline.

Product Area A: Team3
Team3 has four team members. The team is a dedicated DevOps team that takes care of helping the teams on Product A and B to create and maintain their continuous integration and deployment pipelines. Because both products are focused on the cloud, the continuous deployment approach is a central point for the releases. The Team3 does not follow a strict structuring like the other teams. Instead, it consists of a development lead and three engineers. They are structured in a dedicated team to enable synergies between the different products and to ensure knowledge exchange.

Product Area B: Team5
Product Area B is focused on logging related products. First, Team5 of Product D is discussed, because Product C by Team4 is building up on this work.
The Team5 has five team members, including three developers, a Scrum Master that is also doing part-time development, and a PO that is also the Development Manager of Product Area B. Team5 is developing a log management solution together with one team from Gliwice (Poland) and Montreal (Canada) each. The project focuses on retrieval and collection of logs generated by products of the organization and making these logs searchable and analyzable.

Product Area B: Team4
Team4 has seven team members, including six developers and one PO. The Team4 does not have a Scrum Master. Team4 is developing a platform that enables analysis of software logs and reaction to events in the logs. Their Product C is building onto the solution

of Product D and adds more advanced features to the product. Even though this setup of the single Team4 developing Product C does not match the large-scale definition from Section 2.3.1, interviewees were sampled from Team4 because of their close collaboration and high demand for coordination with the teams of Product D.

Product Area C: Team6
Team6 has twelve team members, including a Scrum Master, a PO, an Enterprise Architect, a dedicated Quality Assurance member, a Technical Writer, and a development team of seven people. In total, seven teams distributed across several continents are collaborating on the development of Product E. Product E is a software that automates the deployment and operation of other products of the case organization. The teams of Product E are generally following a Scrum setup. They have implemented a Scrum-of-Scrums meeting and several other practices like the **CO-9**: BUG TRIAGE MEETING or the **M-4**: FOLLOW THE SUN practice to deal with this large setup.

4.3. Findings on Recurring Concerns

This section contains the results of the first open part of the interviews, where we asked participants to identify their top three concerns regarding the methodology and coordination in large-scale agile software development. Most of them identified three concerns and the following list contains all the named concerns. Further, the list also contains concerns that have been identified by observation, in meetings, and in archival data. Even though we explicitly asked for concerns of large-scale agile methodology and coordination, some participants raised concerns from other categories as well. We decided to document them anyway to depict a holistic picture of the concerns at the case organization.

- **C-101**: *How to keep the team motivated despite frequent, severe changes in requirements?* Even though agile values exhort people to embrace change, in the case organization there was still a lot of frustration due to frequent changes in project requirements and scope. Especially in Team1, this concern was relevant for the Scrum Master and Product Owner because in the development work they faced several serious changes in requirements. This concern is categorized as a team level concern of 'Culture & Mindset'.
- **C-102**: *How to deal with corporate hierarchies and salary structures?* Interviewee M1 mentioned he observes that "[...] in traditional corporate structures, which are structured hierarchically, agility reaches its limits." Even though agile teams are structured democratically, in the larger organization "[...] often there are limitations at some point that tell you 'we do this differently here'. You just have to go up far enough, and then there are always some policies or areas where you can't get ahead." Interviewee SM3 also described, that the hierarchical structure in the case organization

sometimes intimidates junior people to raise their concerns against senior colleagues. This concern is categorized as a 'Methodology' concern on all levels.

- **C-103**: *How to deal with frustration by change despite using agile methods?* Several interviewees described, that even though the case organization is heavily engaging in running projects using agile methodologies, people in the teams still get frustrated by change in requirements and scope of projects. SA1 described, that Scrum was taken as the granted way to be more agile by many people. But dealing with the actual change still was a problem for them. Interviewee SA2 linked this concern to evolve out of **C-6**: *How to deal with incorrect practices of agile development?*. This concern is categorized as a 'Culture & Mindset' concern at team level.

- **C-104**: *How to deal with highly ambiguous tasks in agile methods?* This concern describes problems with ambiguous tasks, such as software architecture, in combination with agile methods. SA1 explained "[...] that's so difficult – because you don't know how to measure or estimate those things – to put it in a Scrum setup." Interviewee PO2 added that "[...] if those requirements are not very clear, it becomes a bit difficult to actually prioritize any other feature above those tasks." SA1 mentioned for which tasks this concern is relevant: "[...] when you are looking into "how" and "what", then it's so difficult to create a user story." This concern is categorized as a team level concern of 'Project Management'.

- **C-105**: *How to deal with requirements coming from different sides?* Interviewee SM1 described, that it is "[...] a problem in big organizations that you have requirements coming from left and right, from any random person at any random time." The problem of this concern is the multitude of stakeholders and communication channels, from which requirements can reach the PO and the development team. This concern is categorized as a team level concern of 'Communication & Coordination'.

- **C-106**: *How to deal with lack of time to work with the Product Owner to organize the backlog?* This is a concern for Scrum Masters, that describes coordination problems regarding collaboration with the PO. Because the PO often is very busy, and is working between the two sides of business and development, it was expressed by Scrum Masters that they struggle to find time slots to organize the backlog together with the PO. This concern was enforced by the fact that some Scrum Masters were doing this role only part-time. This concern is categorized as a team level concern of 'Communication & Coordination'.

- **C-107**: *How to deal with slow reactions of other teams or people in case of dependencies?* In case of unforeseen dependencies during a sprint, it is a concern for several interviewees that other teams or individuals in the organization often take a long time to reply to inquiries. In the meantime the work in the ongoing sprint is often blocked. This concern is categorized as an organization level concern of 'Communication & Coordination'.

- **C-108**: *How to deal with different learning speeds of team members?* Scrum Master SM1 expressed the concern, that due to the high pressure regarding approaching deadlines, the team was facing issues at dealing with different working and learning speeds of individual developers. It is hard for the Scrum Master to balance the desire of "slower" people to learn new technologies, and the need of the team to finish critical tasks in time and thus assign them to already experienced team members. SM1 found it difficult to ensure a certain balance in this case. This concern is categorized as a team level concern of 'Methodology'.

- **C-109**: *How to ensure that quality requirements are prioritized in development work?* Interviewee PO2 struggled with prioritization of quality requirements. Especially, because the stakeholders of their project were only focusing on features being finished as soon as possible. Due to this, the team had built up a sizable technical debt, that cost a lot of time to fix and that lowered the members' motivation. This concern is categorized as a 'Software Architecture' concern at program level.

- **C-110**: *How to deal with increasing complexity of systems based on micro-service architecture?* Interviewee D1 mentioned the concern, that, due to the distribution of development across many projects and teams, the emerging micro-service landscape is increasingly complex to handle. "Our main focus here is on coordinating the distributed systems", said D1. The micro-service architecture requires additional coordination between teams and integration between components. This concern is categorized as an organization level concern of 'Software Architecture'.

- **C-111**: *How to keep the team focused on the larger context and project goals?* Interviewee D1 described situations where the development team Team4 had trouble focusing on the larger goal of their project. The team was very busy doing patches and security related tasks, and the usage of the Kanban methodology in this situation led to them thinking only in 'tickets'. The switch to Scrum improved on this, but did not resolve the concern completely. This concern is categorized as a 'Methodology' concern at the team level.

- **C-112**: *How to avoid developing solutions multiple times in the organization?* The case organization had a problem with redundant development work. D1 mentioned that "[...] it's a scaling problem with large companies, that you just don't know what already exists." Therefore, there have been cases of solutions being developed, that already existed elsewhere in the organization. This concern is categorized as an organization level concern of 'Enterprise Architecture'.

- **C-113**: *How to ensure acceptance of Product Owner and Scrum Master by the team?* The manager M2 described that in the past he faced issues with the acceptance of the Product Owner and Scrum Master by the development team. The reason for this was that, though both of them were sophisticated agile practitioners, their missing technical expertise in the area of the project led to disrespectful behaviour by members

of the development team. This concern is categorized as a 'Methodology' concern at the team level.

- **C-114**: *How to deal with urgent bugfix requests?* Interviewees M2 and D2 described situations, in which the development teams had to react very quickly to urgent issue reports. Both interviewees described, that based on contracts their teams had to react to reported bugs in a certain period of time. This lead to the concern of how to arrange this with the general Scrum sprint setup of their teams. This concern is categorized as a program level concern of 'Methodology'.

- **C-115**: *How to take decisions in multiple-team setups?* In multiple-team setups, there has to be one person that can take decisions in deadlock situations. Interviewee M2 described this concern. M2 added, that neither one of the POs nor the Product Manager are the right person to take such decisions. The concern of M2 was to whether make those decisions personally, or to appoint someone to be able to take decisions that affect multiple teams. This concern is categorized as a program level concern of 'Methodology'.

- **C-116**: *How to deal with an existing development team before requirements are existing?* In large organizations, teams often already exist before their project is actually started. This can be due to a previous project of the team being finished, or because staffing has been started before the project is set up. In case of Team1 the latter was the case. The team was created and set up, before the project they were supposed to work on was completely defined. The PO3 mentioned the concern, that this lead to incomplete requirements and changing scope of the project. Ultimately, the initial project of Team1 was even merged into a much bigger project, because the initial project definition was not sophisticated enough. This concern is categorized as a team level concern of 'Communication & Coordination'.

- **C-117**: *How to deal with demo driven development?* This concern was raised by interviewee PO3. He mentioned that due to a demo that has been scheduled in a very early stage of the project of Team1, the development focused only on relevant topics for this demo. This led to a neglection of important project-specific topics, such as a thorough software architecture. The demo was scheduled by a board member, so the team could not help themselves and had to develop demo driven. This concern is categorized as a program level concern of 'Communication & Coordination'.

- **C-118**: *How to deal with resistance to introduction of agile methods?* This concern describes the resistance of team members to the introduction of agile methodologies. Interviewee PO1 raised this concern in the specific context of a research-heavy project, where team members objected the applicability of agile methodologies to their research work. This concern is categorized as a team level concern of 'Culture & Mindset'.

- **C-119**: *How to deal with issues that interrupt the sprint?* This concern describes the problem of handling tasks, that are coming in during a running sprint, and require an action by the development team. Interviewee PO1 said: "One of the main concerns with everything in general development is unexpected tasks or things that were not planned for the sprint." Even though in Scrum the Scrum Master should shield the team from outside influence during a sprint, this cannot always be achieved. E.g., for tasks that require immediate action of a technical person, like a developer. This concern is categorized as a team level concern of 'Methodology'.

- **C-120**: *How to coordinate work across multiple time zones?* Interviewee SM2 mentioned that "[...] the times and working across time zones is the biggest challenge in working in this distributed team." The main reason for working across multiple time zones being a concern is the short overlap times between the time zones to coordinate all the work. In the case organization the working time overlap between the teams in Montreal and those in Munich was especially short, with only two hours each day. Further, interviewee AC1 added to this concern that this "[...] has an impact on the work live balance". It forces teams "[...] to meet in strange hours that are not really typical for their time zone." This concern is categorized as a team and program level concern of both 'Communication & Coordination'.

- **C-121**: *How to deal with not being able to physically sit together in distributed teams?* Teams that are co-located can sit together physically in front of their whiteboards to have meetings such as the Daily Scrum and Sprint Plannings. They can concentrate on the work they are doing. In contrast, distributed teams cannot sit together. They have to rely on technology to communicate and replicate the board they are using in the meeting. Their way of holding meetings is restricted by the possibilities of the technology. This concern is also linked to **C-138**: *How to deal with distractions in online meetings?*. This concern is categorized as a team level concern of 'Communication & Coordination'.

- **C-122**: *How to deal with unexpected dependencies?* This concern describes situations, where teams discover unexpected or unforeseeable dependencies during a running sprint. In a particular case described by interviewee SM2, a developer was absent for a period of time. Other team members continued their work and reached a point, where they could not progress anymore without the task of the absent person being done. The concern can appear both inside teams and between teams. This concern is categorized as a concern of 'Communication & Coordination' on all scaling levels.

- **C-123**: *How to deal with sub grouping due to geographical distribution?* Interviewee SM2 described that "[...] people group into small sub-teams at the different locations, and there is a preference to whatever your team mates say [...]". This makes it hard to reach location overarching agreements on controversial topics throughout teams. This concern is categorized as a team level concern of 'Geographical Distribution'.

- **C-124**: *How to implement scaled agile methodologies?* The participant AC1 mentioned being concerned with actually applying agile values and methodologies in larger scale. Especially, the continuous improvement of work processes and running retrospectives on larger scale is a concern to AC1. Inter-team coordination and communication processes on higher levels are often not as straight forward as on team level and are therefore hard to grasp and improve by team retrospectives. This concern is categorized as a 'Methodology' concern across all scaling levels.
- **C-125**: *How to plan work and track progress?* While agile frameworks provide methods to plan work in individual teams, they do not provide guidance regarding multiple teams. In the case organization, the concern was raised on how to actually plan work across multiple teams and how to check whether the project is progressing in the right direction. This concern is categorized as a 'Methodology' concern on all scaling levels.
- **C-126**: *How to deal with lack of social binding between teams due to geographical distribution?* In large, distributed teams or in projects with teams distributed across several locations, participants face the concern that members do not feel a social binding or a warm atmosphere between others working in different locations. The reason for this are missing face to face communication, and the lack of any social interaction besides work whatsoever. Interviewee D4 described that there is a clear lack of the feeling among people of Product E, that they belong to the same team. This concern is categorized as a 'Geographical Distribution' concern on team, program and organizational levels.
- **C-127**: *How to meet release dates?* In the case organization projects are often started before their purpose and requirements have been finally decided. This makes it hard to actually define and meet deadlines for the project. This is due to the reason, that the case organization is mostly developing standard software. There is no direct customer driving or pressurizing the development of the product. This concern is loosely linked to **C-116**: *How to deal with an existing development team before requirements are existing?*. This concern is categorized as a program level concern of 'Methodology'.
- **C-128**: *How to introduce agile practices sustainable?* Development Managers expressed that while the introduction of agile methodologies was not that big of a problem in their product areas, ensuring that they are adopted sustainable and are applied consistently is a concern to them. They observed that people tend to fall back into old behavior over time. It is a concern to them, to ensure correct application of agile methods over a longer period of time. This concern is categorized as a program level concern of 'Methodology'.
- **C-129**: *How to avoid wasting time on technical exploration?* Interviewee M1 faced the concern, that in the case organization time was often wasted on unnecessary tech-

nical explorations, like proof of concepts and spikes. One reason for this was the missing clear definition of the Software Architect role, and how this role interacts with the development team. This concern is categorized as a 'Methodology' concern at team level.

- **C-130**: *How to deal with people that stick with their way of doing things?* People tend to stick with work processes and tools that they have been using in the past. Consequently, it is often a concern to introduce new processes or new tools for certain processes. Interviewee SA2 mentioned the example of documentation. People in the case organization are used to document and search for documentation on the corporate wiki. For some projects, according to SA2, this might not be the best way to maintain the documentation. However, SA2 expressed that it is very hard to get people to take a different approach, even if it might be better suited. This concern is categorized as a 'Culture & Mindset' concern at team level.

- **C-131**: *How to deal with political decisions?* Several interviewees described situations, where discussions and decisions turned political and were devoid of a rational basis. Interviewee SA2 said: "It's quite often that some things are simply not done because someone doesn't like someone else. That's the politics." Further, interviewee PO1 connected this concern to higher hierarchical levels: "Usually it is not like my solution is technically better [...], usually the managers are fighting for their solution. I believe this is on higher levels and it is related to big companies." This concern is categorized as a 'Culture & Mindset' concern on all scaling levels.

- **C-132**: *How to increase project visibility in the organization?* Because the case organization is large and distributed, not everyone can be aware of which projects and products are currently being developed. Interviewee M1 was struggling with incorporating internal marketing effort, in order to make other members of the organization aware of the product they were developing. This concern is categorized as a 'Communication & Coordination' concern on program level.

- **C-133**: *How to deal with increased demand of status updates in agile methods?* Interviewee SA1 mentioned that, especially in his architecture work, the Daily Scrum meeting is pressuring people to come up with artificial status updates. In particular, if the work takes longer than expected. This reduces the value of these meetings and is also linked to **C-134**: *How to deal with inefficient coordination meetings?*. This concern is categorized as a 'Methodology' concern at the team level.

- **C-134**: *How to deal with inefficient coordination meetings?* This concern was raised by interviewee SA1: "Like I said, I strongly believe that most of the meetings are inefficient." This is on the one hand due to **C-133**: *How to deal with increased demand of status updates in agile methods?*, and on the other hand because of **C-138**: *How to deal with distractions in online meetings?*. Most of the meetings are not relevant to all of their participants. Thus, people get distracted and the value of the meetings decreases

for all attendees. This concern is categorized as a 'Communication & Coordination' concern at the team level.

- **C-135**: *How to align teams from independent projects to integrate their products?* The case organization did acquire over 25 other companies during the past ten years. This acquisitions come with additional efforts of integrating their products with the existing product line-up of the case organization. Specifically, Team1 had to integrate the product of a company, that was acquired in 2017, into the project they were working on during the time of this study. This concern is categorized as a enterprise level concern of 'Communication & Coordination'.

- **C-136**: *How to avoid building up technical debt due to fast iteration?* This concern was described by participants from the teams Team2 and Team4. Because of pressure from the stakeholders and too short iteration cycles, both teams built up technical debt in the past. It cost them considerable time and effort to fix it afterwards. This concern is categorized as a 'Methodology' concern at the team level.

- **C-137**: *How to balance shielding of the developers and giving them enough project context?* This is a PO concern described by PO2 and PO3. It is the responsibility of the PO to balance the amount of information that passes from the business side to the development side, and vice versa. It is further also the responsibility of the PO to recognize which concerns from which stakeholders are actually relevant to the project. This is linked to **C-105**: *How to deal with requirements coming from different sides?*. However, also interviewee D1 raised this concern. For developers, the missing information that the PO "filtered" out can often lead to a loss of context. It is therefore also a concern for developers to ensure that they get enough contextual information for their work. This concern is categorized as a 'Methodology' concern at the team level.

- **C-138**: *How to deal with distractions in online meetings?* Additionally to **C-134**: *How to deal with inefficient coordination meetings?*, people tend to get distracted in online meetings even more than in face-to-face meetings. Interviewee SM2 mentioned that "[...] you can feel that they [do] not really completely participate in the meeting, because you don't have this under control, right." Interviewee PO1 also added a technical dimension to it: "[...] if you do that on a Skype-Call, like, it is normal that people get disconnected." This concern is categorized as a 'Communication & Coordination' concern at the team level.

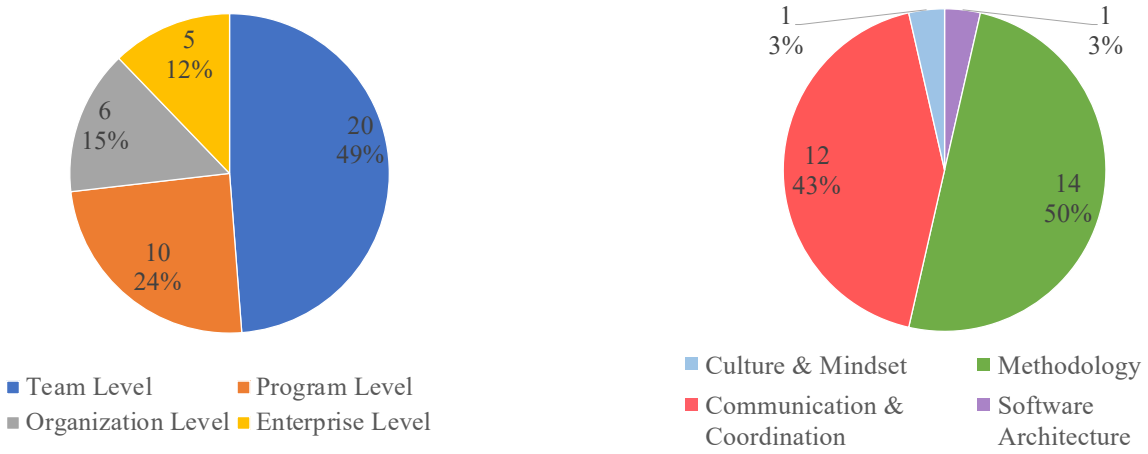
- **C-139**: *How to deal with lacking knowledge of another team's activities?* Due to the distribution of teams, and also due to the big number of teams of Product Area C, the concern occurred that individuals or even whole teams did not exactly know what their peer teams were actually working on at the moment. This has led to coordination problems and even to unnecessary efforts and wrong development work. This concern is categorized as a 'Communication & Coordination' concern at team level.

In sum 39 concerns have been identified during the interviews and observations at the case organization. 13 identified concerns are categorized as 'Communication & Coordination' and 15 identified concerns are categorized as 'Methodology' concerns. Besides those two categories on which this thesis focuses, we also identified some concerns from other categories: 5 concerns of category 'Culture & Mindset', 1 concern of category 'Project Management', 2 concerns of category 'Software Architecture', 1 concern of category 'Enterprise Architecture', and 2 concerns of category 'Geographical Distribution'.

Removing and Merging Duplicated Concerns

The interview participants were asked to identify their most pressing concerns *before* they were shown the list of already identified concerns. Therefore, some duplicates emerged throughout the course of the study. The duplicates will be merged with existing concerns in this section. We will also show justification why (allegedly) similar concerns have not been merged. In this section a citation is placed behind mentions of concerns that originate from literature to make the process better understandable.

First, the concerns **C-101**: *How to keep the team motivated despite frequent, severe changes in requirements?* and **C-103**: *How to deal with frustration by change despite using agile methods?* are related, as they both cover the frustration and loss of motivation that results out of drastic changes in a running project. Therefore, **C-103** is removed and all links to it are mapped to **C-101**. However, the concerns are not merged with the similar concern **C-7**: *How to deal with doubts in people about changes?* [75], because **C-7** refers to the doubts regarding incoming new changes and **C-101** refers to the frustration that emerges after the changes have been accepted for implementation. The concern **C-104**: *How to deal with highly ambiguous tasks in agile methods?* is merged into the concern **C-73**: *How to deal with decreased predictability?* [75], because they both describe the ambiguity and the resulting problems in dealing with the planning and prediction of work effort. Next, the concern **C-107**: *How to deal with slow reactions of other teams or people in case of dependencies?* is similar to the concern **C-21**: *How to manage dependencies to other existing environments?* [75]. However, the concerns are not merged, because **C-21** refers to dependencies from the architectural perspective, but **C-107** refers to any kind of dependencies that might occur from the development team's perspective such as requesting knowledge or unavailable colleagues. The concerns **C-109**: *How to ensure that quality requirements are prioritized in development work?* and **C-8**: *How to ensure that non-functional requirements are considered by the development team?* [75] are merged, since quality requirements are non-functional requirements [17]. All references to **C-109** are mapped to **C-8**. **C-112**: *How to avoid developing solutions multiple times in the organization?* and **C-34**: *How to ensure the reuse of enterprise assets?* [75] are merged, because they both describe the same concern from different perspectives. The first one describes to avoid developing something that already exists and the second one describes to make sure that something that already exists is reused. Therefore, all references to **C-112** are mapped to **C-34**. The concerns **C-131**: *How to deal with political decisions?* and **C-65**: *How to deal with office politics?* [75] are merged, since **C-131** is just a rephrasing of **C-65**. Hence,



(a) Scaling levels

(b) Categories

Figure 4.3.: Overview of the scaling levels and categories of the identified concerns

all references to **C-131** are mapped to **C-65**. **C-119**: *How to deal with issues that interrupt the sprint?* and **C-41**: *How to deal with unplanned requirements and risks?* [75] are similar but are not merged, because **C-119** also covers unplanned issues that are neither requirements nor risks (e.g., organizational issues). The concern **C-118**: *How to deal with resistance to introduction of agile methods?* is merged with the concern **C-46**: *How to deal with closed mindedness?* [75], since the resistance to introduction of agile methods is related to closed mindedness of individuals. All references to **C-118** are mapped to **C-46**. Additionally, references to **C-130**: *How to deal with people that stick with their way of doing things?* are also merged and mapped to **C-46**. The concerns **C-126**: *How to deal with lack of social binding between teams due to geographical distribution?* and **C-123**: *How to deal with sub grouping due to geographical distribution?* describe the same phenomenon as **C-32**: *How to deal with lacking team cohesion at different locations?* [75]. Therefore, the three concerns are merged and all references to **C-126** or **C-123** are mapped to **C-32**. The concern **C-114**: *How to deal with urgent bugfix requests?* is already mentioned in [62] as **C-95**: *How to rapidly deliver a necessary patch?*. All references to **C-114** are mapped to **C-95**. **C-114** is removed. Finally, the concern **C-139**: *How to deal with lacking knowledge of another team's activities?* already exists in [62] and was assigned the identifier **C-83**. Thus, **C-139** is removed and all references to it are mapped to **C-83**.

After removing and merging all duplicates, 28 new concerns have been identified in this study at the industry partner. 12 of them are categorized as 'Communication & Coordination', and 14 are categorized as 'Methodology'. Figure 4.3 gives an overview about the scaling levels and categories of the newly identified concerns after the merging of duplicates. The absolute numbers shown in Figure 4.3a are higher than 28 because five concerns applied to more than one scaling level.

Identification of Recurring Concerns

In the last part of the interviews, we presented the list of already identified concerns to the participants. This list contained all concerns from previous interviews as well as concerns identified in literature. We asked them to identify those concerns on the list that they are also facing in their work in their current role. For each of the concerns on this list, the following Tables 4.3, 4.4, and 4.5 show how many of the practitioners stated that they are also having the respective concern in their work. The percentage values are all calculated based on the number of 15 interviews.

Figure 4.4 shows all interviewed stakeholder-categories (Developer (Dev), Scrum Master (SM), Product Owner (PO), Development Manager (DM), and Agile Consultant(AC)) and all newly identified concerns after the merge and removal of duplicates. For each concern a connection to a stakeholder-category is drawn if at least half of the interviewees of this stakeholder-category identified this concern as being relevant to them. The *rule of three* could not be applied meaningfully in this case because only the Developer stakeholder-category had more than three participants in the conducted interview series. We find that while many of the identified concerns are relevant to multiple stakeholder-categories, the observed and interviewed practitioners did not actively apply practices against all their relevant concerns. This indicates that although a concern is relevant to a stakeholder-category, they do not necessarily act against it. The formal definition of the LSADPL allows for multiple stakeholders having the same concern [74]. However, for future work it might be useful to differentiate between *relevant* concerns for a stakeholder and concerns that are actually *actively addressed* by the stakeholder. Additionally, in Figure 4.4 the concerns that have been identified as being relevant by three or more participants from different teams are outlined with a black border. The figure shows that besides concerns **C-117** and **C-129** all newly identified concerns are relevant for three or more of the six teams.

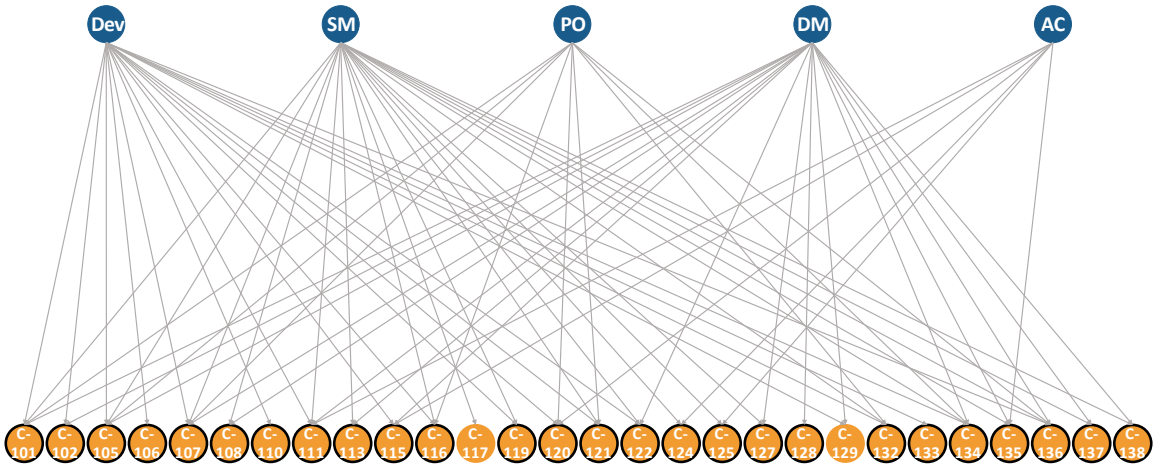


Figure 4.4.: Visualization of which concerns are relevant to which stakeholder-category

ID	Name	Category	Source	#	Percentage
C-1	How to coordinate multiple agile teams that work on the same product?	Communication & Coordination	[75]	10	67%
C-6	How to deal with incorrect practices of agile development?	Methodology	[75]	7	47%
C-16	How to establish self-organization?	Communication & Coordination	[75]	5	33%
C-20	How to facilitate communication between agile teams and other teams using traditional practices?	Communication & Coordination	[75]	2	13%
C-44	How to deal with communication gaps with stakeholders?	Communication & Coordination	[75]	11	73%
C-49	How to deal with increased efforts by establishing inter-team communication?	Communication & Coordination	[75]	7	47%
C-59	How to establish a common understanding of agile thinking and practices?	Methodology	[75]	11	73%
C-63	How to explain requirements to stakeholders?	Communication & Coordination	[75]	5	33%
C-75	How to form and manage autonomous teams?	Communication & Coordination	[75]	7	47%
C-78	How to build an effective coaching model?	Methodology	[75]	3	20%
C-79	How to synchronize sprints in the large-scale agile development program?	Communication & Coordination	[75]	5	33%
C-80	How to deal with a competing concept deadlock?	Communication & Coordination	[62]	8	53%
C-81	How to deal with missing communication of decommitment?	Communication & Coordination	[62]	5	33%
C-82	How to deal with unclear mutual expectations?	Communication & Coordination	[62]	6	40%
C-83	How to deal with lacking knowledge of another team's activities?	Communication & Coordination	[62]	10	67%
C-84	How to deal with unknown dependencies between teams?	Communication & Coordination	[62]	9	60%
C-85	How to deal with increase in geographic dispersion?	Communication & Coordination	[62]	5	33%
C-86	How to deal with corruption of a shared codebase?	Methodology	[62]	2	13%
C-87	How to deal with a work item spanning across teams?	Communication & Coordination	[62]	5	33%
C-88	How to deal with unclear work items?	Communication & Coordination	[62]	9	60%
C-89	How to deal with unclear usage of a new development framework?	Communication & Coordination	[62]	5	33%
C-90	How to deal with major testing failures of new feature?	Communication & Coordination	[62]	5	33%

Table 4.3.: List of occurrences of concerns from the last part of the interview, absolute and relative numbers (based on 15 interviews) shown in table

ID	Name	Category	Source	#	Percentage
C-91	How to deal with new cross-team feature originating from one team?	Communication & Coordination	[62]	2	13%
C-92	How to deal with priority conflict within takt?	Communication & Coordination	[62]	6	40%
C-93	How to deal with assumption mismatch?	Communication & Coordination	[62]	8	53%
C-94	How to deal with late delivery of needed functionality?	Communication & Coordination	[62]	5	33%
C-95	How to rapidly deliver a necessary patch?	Methodology	[62]	6	40%
C-96	How to deal with unresolved prioritization of a topic?	Communication & Coordination	[62]	6	40%
C-97	How to deal with a cross-team item facing asymmetric team knowledge?	Communication & Coordination	[62]	5	33%
C-98	How to deal with recognition of a reuse possibility?	Communication & Coordination	[62]	6	40%
C-99	How to deal with discovery of redundancies?	Communication & Coordination	[62]	9	60%
C-100	How to deal with division of knowledge within and between teams?	Knowledge Management	[10]	9	60%
C-101	How to keep the team motivated despite frequent, severe changes in requirements?	Culture & Mindset	*new*	13	87%
C-102	How to deal with corporate hierarchies and salary structures?	Methodology	*new*	7	47%
C-105	How to deal with requirements coming from different sides?	Communication & Coordination	*new*	13	87%
C-106	How to deal with lack of time to work with the Product Owner to organize the backlog?	Communication & Coordination	*new*	6	40%
C-107	How to deal with slow reactions of other teams or people in case of dependencies?	Communication & Coordination	*new*	10	67%
C-108	How to deal with different learning speeds of team members?	Methodology	*new*	4	27%
C-110	How to deal with increasing complexity of systems based on micro-service architecture?	Software Architecture	*new*	5	33%
C-111	How to keep the team focused on the larger context and project goals?	Methodology	*new*	11	73%
C-113	How to ensure acceptance of Product Owner and Scrum Master by the team?	Methodology	*new*	6	40%
C-115	How to take decisions in multiple-team setups?	Methodology	*new*	9	60%
C-116	How to deal with an existing development team before requirements are existing?	Communication & Coordination	*new*	7	47%
C-117	How to deal with demo driven development?	Communication & Coordination	*new*	3	20%

Table 4.4.: List of occurrences of concerns (Table 4.3 continued)

ID	Name	Category	Source	#	Percentage
C-119	How to deal with issues that interrupt the sprint?	Methodology	*new*	7	47%
C-120	How to coordinate work across multiple time zones?	Communication & Coordination	*new*	8	53%
C-121	How to deal with not being able to physically sit together in distributed teams?	Communication & Coordination	*new*	9	60%
C-122	How to deal with unexpected dependencies?	Communication & Coordination	*new*	7	47%
C-124	How to implement scaled agile methodologies?	Methodology	*new*	7	47%
C-125	How to plan work and track progress?	Methodology	*new*	6	40%
C-127	How to meet release dates?	Methodology	*new*	7	47%
C-128	How to introduce agile practices sustainable?	Methodology	*new*	4	27%
C-129	How to avoid wasting time on technical exploration?	Methodology	*new*	3	20%
C-132	How to increase project visibility in the organization?	Communication & Coordination	*new*	6	40%
C-133	How to deal with increased demand of status updates in agile methods?	Methodology	*new*	7	47%
C-134	How to deal with inefficient coordination meetings?	Communication & Coordination	*new*	10	67%
C-135	How to align teams from independent projects to integrate their products?	Communication & Coordination	*new*	8	53%
C-136	How to avoid building up technical debt due to fast iteration?	Methodology	*new*	8	53%
C-137	How to balance shielding of the developers and giving them enough project context?	Methodology	*new*	7	47%
C-138	How to deal with distractions in online meetings?	Communication & Coordination	*new*	8	53%
C-140	How to deal with external developers?	Communication & Coordination	[35]	6	40%
C-141	How to coordinate multi-vendor teams?	Communication & Coordination	[35]	5	33%
C-142	How to deal with decisions on higher levels reaching lower levels?	Communication & Coordination	[35]	12	80%
C-143	How to deal with missing understanding of roles?	Methodology	[35]	7	47%
C-144	How to deal with increased number of coordination meetings?	Communication & Coordination	[35]	7	47%

Table 4.5.: List of occurrences of concerns (Table 4.4 continued)

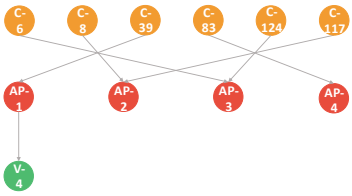
We created a table of all newly identified concerns after the merge and removal of duplicates, including their description, categorization, scaling level, and interview of origin. It can be found in Appendix B.1.

4.4. Findings on Good and Bad Practices

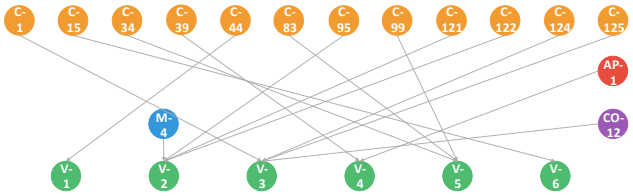
During the five months period of this study we observed and identified 23 coordination pattern candidates, 9 methodology pattern candidates, 6 viewpoint pattern candidates, 4 anti-pattern candidates, and 2 principle candidates. In sum we documented 42 pattern candidates. The observed principle and pattern candidates are:

CO-1: JOINT DAILY SCRUM MEETING
CO-2: REFINEMENT MEETING
CO-3: DEVCORNER
CO-4: INTEGRATION COORDINATION GROUP
CO-5: PROJECT DEBRIEFING
CO-6: REPRESENTATIVE EXCHANGE
CO-7: PROJECT STATUS PROTOCOL
CO-8: AREA RETROSPECTIVE
CO-9: BUG TRIAGE MEETING
CO-10: CROSS TEAM PEER REVIEW
CO-11: DISTRIBUTED COMPONENT LEADS
CO-12: MILESTONE PLANNING MEETING
CO-13: COFFEE CORNER MEETING
CO-14: LUNCH TALK
CO-15: OPEN WORK AREA
CO-16: INSTANT MESSAGING
CO-17: ONLINE FORUM
CO-18: AD HOC COMMUNICATION
CO-19: DAILY STANDUP MEETING
CO-20: SPRINT RETROSPECTIVE MEETING
CO-21: SPRINT REVIEW MEETING
CO-22: SPRINT PLANNING MEETING
CO-23: ISSUE TRACKING TOOL
M-1: MONO-REPO
M-2: MIXED SPRINT
M-3: ASSIGNING RIGHTS
M-4: FOLLOW THE SUN
M-5: SHIPCAPTAIN
M-6: SPRINT ZERO
M-7: DEDICATED PERSON TO DEAL WITH ANNOYMENTS
M-8: REQUIREMENT SEPARATION
M-9: PAIR PROGRAMMING
V-1: ROADMAP
V-2: TASK DEPENDENCY MAPPING
V-3: MILESTONE PLANNING BOARD
V-4: SAILBOAT RETROSPECTIVE
V-5: TEAM HOMEPAGE
V-6: PERSONA
AP-1: RANTROSPECTIVE
AP-2: DEMO DRIVEN DEVELOPMENT
AP-3: DON'T USE AGILE AS MAGIC BULLET
AP-4: TOO HIGH-LEVEL SCRUM OF SCRUMS
P-1: PREREQUISITES TO FORM AUTONOMOUS TEAMS
P-2: SPREAD KNOWLEDGE

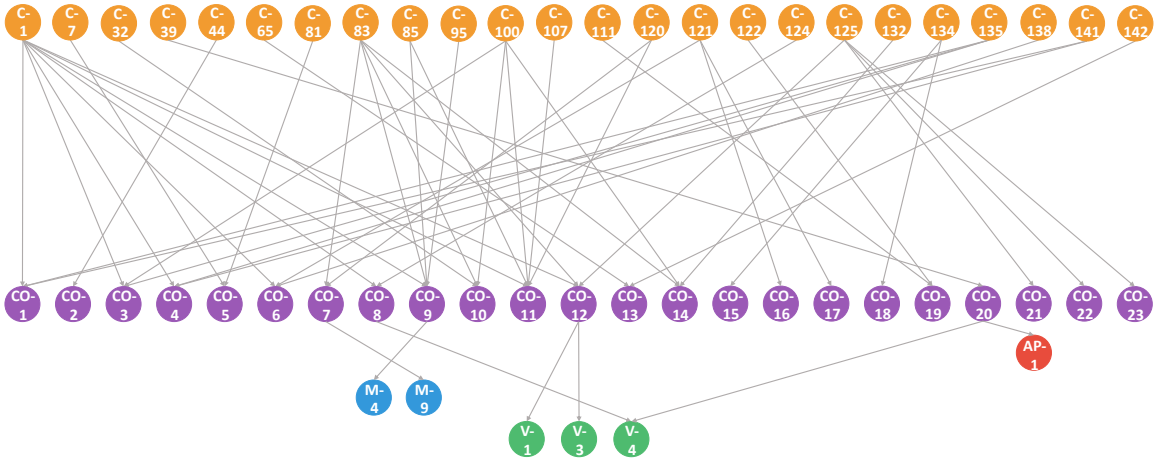
Table 4.6.: Overview of all identified principle and pattern candidates



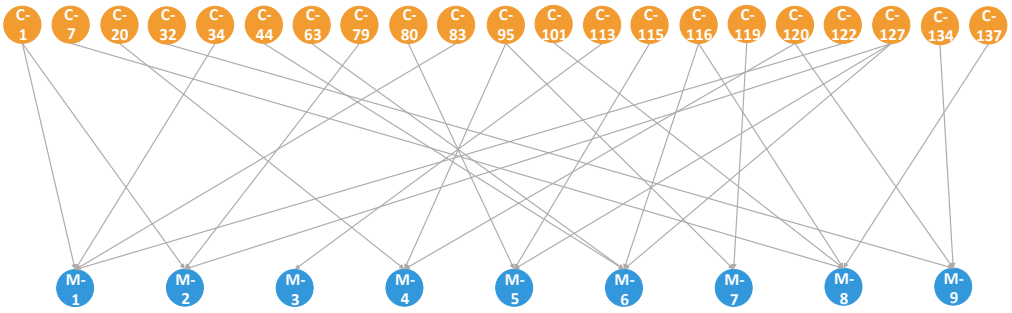
(a) Anti-Pattern candidates



(b) Viewpoint-Pattern candidates



(c) Coordination-Pattern candidates



(d) Methodology-Pattern candidates

Figure 4.5.: Visualization of which concerns are addressed by the identified practices of each category

A complete documentation of all identified pattern candidates, including all their important attributes, can be found in the Appendix C. Figure 4.5 shows which concerns are addressed by the identified good and bad practices. The figure has been split on practice category to make it easier to track connections. Figure 4.6 shows the practices that have been mentioned and applied by each team. Additionally to the teams, Figure 4.6 also shows a row for the interviewee AC1. This is because AC1 cannot be linked to one or more teams but is working with all the teams on a need basis. AC1 mentioned some practices that we could not observe at the six teams during the study. Those are the pattern candidates that are only linked to AC1 in Figure 4.6.

Following, this part of the thesis presents and discusses five pattern candidates in more detail. The selection was based on frequency of observation. The selected practices where among the most often used ones in their categories (and are not Scrum meetings or office wide practices).

	CO-1	CO-2	CO-3	CO-4	CO-5	CO-6	CO-7	CO-8	CO-9	CO-10	CO-11	CO-12	CO-13	CO-14	CO-15	CO-16	CO-17	CO-18	CO-19	CO-20	CO-21	CO-22	CO-23
Team1	X	X	X	X	X								X	X	X	X	X	X	X	X	X	X	X
Team2													X	X	X	X	X	X	X	X	X	X	X
Team3													X	X	X	X	X	X	X	X	X	X	X
Team4				X		X							X	X	X	X	X	X	X	X	X	X	X
Team5						X	X						X	X	X	X	X	X	X	X	X	X	X
Team6							X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
AC1								X				X											

	M-1	M-2	M-3	M-4	M-5	M-6	M-7	M-8	M-9	V-1	V-2	V-3	V-4	V-5	V-6	AP-1	AP-2	AP-3	AP-4	P-1	P-2
Team1	X		X		X	X		X	X					X	X	X	X	X		X	
Team2		X	X		X					X				X			X	X			
Team3														X							
Team4	X		X		X		X							X							X
Team5			X	X	X	X			X		X			X			X				
Team6				X					X					X	X				X		
AC1				X		X						X	X								

Figure 4.6.: Overview of which practices were applied by which teams

4.4.1. Follow the Sun

Pattern Overview	
ID	M-4
Name	FOLLOW THE SUN
Alias	Dispatcher
Summary	To deal with urgent bug fix requests and customer issues, the Follow the Sun practice helps to guarantee a certain reaction time. The distribution of the teams across multiple time zones, together with the role of a 'Dispatcher', is leveraged to have 24/7 availability for urgent issues.

Example
The initial version of Product E of the case organization has been released half a year ago. Since then, the live running installations of the product have significantly increased. This led to a high inflow of bug reports and urgent customer issues. The company is obliged by contract to react to reported issues in less then two hours. To keep up with this, the project teams set up a 'Dispatcher' role and have a rotating on-call team in each time zone.

Context
The company has to react to issue reports in a given time frame. The number of incoming issues and reports is high. Teams are distributed across multiple time zones.

Problem
C-20: *How to facilitate communication between agile teams and other teams using traditional practices?*
C-95: *How to rapidly deliver a necessary patch?*
C-120: *How to coordinate work across multiple time zones?*

Forces

- Customers can report issues at any given time, all around the globe. The company needs to make sure to be on call all day.
- The protective setup of Scrum and other agile methods during a sprint makes it hard to address urgent issues during a running sprint.

Solution
To achieve very quick reaction times to urgent bug fix requests, set up a '24/7 Team'. This team is distributed across different time-zones, in such a way that 24/7 coverage is ensured by always having one active team. Create a 'Dispatcher' role that takes bugs and reported issues and directly assigns them to development teams or developers – even inside a running sprint. Each time-zone team has one Dispatcher that is responsible for six

hours a day, so 24/7 team availability is achieved by rotation. The Dispatcher has access to the bug database / error tracking system. The Dispatcher has the right to interfere the work of the Scrum development teams and can assign tasks even inside a running sprint. If necessary, the Dispatcher can also release code bypassing the normal review procedure. During the last two hours of the workday, the Dispatcher ensures that all currently running work is handed over to the following Dispatcher and 24/7 team members in the next time zone.

Consequences
Benefits:

- The reaction time to urgent requests is guaranteed to be low around the clock.
- The normal Scrum development setup can continue in parallel without changes.

Liabilities:

- The Scrum sprints and meeting cycle can be disturbed.

See Also

- This practice can be applied together with **V-2**: TASK DEPENDENCY MAPPING to visualize the necessary handovers across time zones.

Other Standards

- A similar pattern is documented in the pattern catalogue by [44] and in [16].

4.4.2. Background on Follow the Sun Practice

The Follow the Sun practice was discussed in the interview with participant M2 under the name 'Dispatcher'. This name is used as an alias in the documentation. The interviewee D2 described that the Follow the Sun practice was actively being used in their project to deal with urgent issue reports and approaching deadlines. When talking about the concern **C-120**: *How to coordinate work across multiple time zones?*, interviewee AC1 mentioned that one of the few benefits of distributing teams across multiple time zones was being able to use the Follow the Sun practice. The Follow the Sun practice leverages the fact that due to the distribution across different time zones, the effectively used working hours around the day can be increased. This is achieved by doing handovers and work transfers between teams in the short time zone overlaps. It allows to be working and to be on call for issues up to 24 hours each day, if enough time zones are involved.
In the Follow the Sun practice the role of the 'Dispatcher' is important. Interviewee M2 described the Dispatcher role in detail. Participant D2 described a similar role being filled by Product Owners in their project. Essentially, the Dispatcher takes care of receiving incoming issues and relaying them to the teams in the currently available time zone. Thus, the role has to follow the sun and is assigned to one person in each time zone. While the

normal development of the project is going on, the Dispatcher can intervene the standard sprint schedule and assign urgent tasks directly to developers or teams. This guarantees a certain reaction time to issues, which the case organization was obliged to by contract in some projects.

4.4.3. Sprint Zero

Pattern Overview	
ID	M-6
Name	SPRINT ZERO
Alias	Phase Null
Summary	To get a common understanding of the project and to align every stakeholder with the requirements, a Sprint Zero is conducted before the development work is started. It is similar to a normal Sprint, however the sprint backlog is filled with tasks on feature prioritization, work planning, and assigning stories to future sprints, rather than producing a deliverable product increment.

Example
Team1 was developing the predecessor of Product A. During the development, the requirements and project scope changed drastically. The Scrum Master of Team1 decided to run a Sprint Zero before the next actual development Sprint to ensure that the development team and the stakeholders are aligned again.

Context
A new project is set up by the company. The stakeholders and development team are about to start their work. Or the requirements of an already running project are changing very drastically.

Problem
C-44: *How to deal with communication gaps with stakeholders?*
C-63: *How to explain requirements to stakeholders?*
C-116: *How to deal with an existing development team before requirements are existing?*
C-127: *How to meet release dates?*

Forces

- In the agile context it is – by definition – hard to define what will be delivered when.
- Stakeholders often do not have a clear understanding of the requirements.
- Stakeholders and the development team often have different views on what is technically possible.

Solution
To get all the stakeholders and the development team aligned on requirements and vision for a new project, conduct a Sprint Zero before actual development work starts. All stakeholders of the project and the development team physically participate in the Sprint Zero. The tasks of the Sprint Zero are focused on definition of requirements, establishing a common vision among all stakeholders, prioritization of features, and general structuring of future work. The outcome is not a deliverable product increment but a thoroughly defined base for starting the actual development work.

Variants
This practice was also described in a workshop-like setting instead of a sprint structure.
A variant, which has been used by Team1, is to also conduct a Sprint Zero in cases when the requirements of the project are changing and the development team needs to be realigned with the other stakeholders.

Consequences
Benefits:

- All stakeholders get a common understanding of requirements.
- All stakeholders understand what is technically doable by the development team.
- Delivery plans and deadlines can be estimated in cooperation and more precisely.

Liabilities:

- Usually it is hard to physically gather all stakeholders for a period of a sprint.
- Danger of being too fixed on the created backlog and prioritization after the Sprint Zero.

See Also

- The Sprint Zero is also mentioned in [72].

4.4.4. Background on Sprint Zero Practice

A Sprint Zero was conducted by Team1 during the period of observation. Interviewee PO3 pointed out that the team was facing the concern **C-116**: *How to deal with an existing development team before requirements are existing?*. The team had been set up during summer 2018 and the requirements of the project since then kept changing extensively. Interviewee M1 described a similar practice, calling it 'Phase Null'. This name is used as an alias for the practice in the documentation. Participant AC1 described a similar practice in form of a kick-off workshop that is conducted before project start.
The Sprint Zero practice describes a phase of bringing together and aligning the various

stakeholders of a project. It is called 'Sprint Zero' because this phase can be structured just like a normal sprint. It can include the Sprint Planning, Sprint Review, and Retrospective meetings. The Sprint Zero is conducted before a project is actually started and before any development work is done. All the stakeholders come together physically in one place. The goal is to establish a common understanding of the requirements and to build a common vision of the project. A Sprint Zero can also be conducted between two normal sprints, which is what we observed at Team1. This can be useful after profound changes have been made to the requirements or the environment of the project. It helps to realign all the stakeholders and to reshape the common vision for the project.

4.4.5. Representative Exchange

Pattern Overview	
ID	CO-6
Name	REPRESENTATIVE EXCHANGE
Alias	
Summary	If multiple teams are working together on the same product, at least one delegate of each team physically participates in Sprint Plannings and Reviews of the other teams respectively.

Example
Team5 is developing their product together with other teams located elsewhere. To make sure all important information is exchanged between the distributed teams, a delegate of each team takes part in the Sprint Planning and Review meetings of the other teams respectively.

Context
Multiple teams working together on one project or product. The teams are not co-located.

Problem
C-1: *How to coordinate multiple agile teams that work on the same product?*
C-121: *How to deal with not being able to physically sit together in distributed teams?*
C-138: *How to deal with distractions in online meetings?*

Forces

- People often get disconnected from remote calls (e.g., via Skype), either physically by losing connection or by getting distracted from other things on the computer.
- Discussions about complex topics are easier to do in person because remote calls are not as interactive.

Solution
To make sure all important information is exchanged between distributed teams, a delegate of each team takes part in the Sprint Planning and Review meetings of the other teams respectively. The delegate can ask in-depth questions at those meetings and can inform his 'home-team' about important decisions and work that is going on in the other team. Further, the delegate can serve as a contact person for the remote team to ask questions about the work of the 'home-team'.

Variants
The Representative Exchange can also be conducted with special groups such as Security and Quality Assurance. In this case, the special group sends a delegate to the different development teams of a project to ensure that their interests are taken into account at the various projects of the organization.

Consequences
Benefits:

- The delegate can bridge the communication gap between the teams.
- The delegate can ask clarifying questions and request information outside of the scope of a remote call.
- The delegate can also provide information about what his own team is doing.

Liabilities:

- It is expensive to allow the delegate to travel to the other team's location.
- The more teams that are collaborating on a project, the more delegates have to be sent out.

4.4.6. Background on Representative Exchange Practice

The Representative Exchange practice was described in two contexts. Interviewee PO1 described that they were using the Representative Exchange between different development teams, to attend the Sprint Plannings and Sprint Review Meetings of their peer teams on the project. Interviewee M1 described the Representative Exchange in the context of integrating representatives from special interest groups, such as Security, Quality Assurance, and Software Architecture. It helps integrating them directly into the development work of the teams. However, as the resources of these groups are limited and the organization can be large, it might not be possible to send a delegate to each development team in the organization.
The Representative Exchange practice focuses on the problem that distributed teams cannot have their meetings physically together. To deal with this situation, the teams of a project can exchange individual delegates to physically attend at least the Sprint Reviews and Sprint Plannings of the other teams respectively. If possible, the delegates could attend even more meetings. This helps establishing a hot wire to the peer teams and ensures all

necessary information is exchanged between the teams. The delegate serves as a connector for teams.

4.4.7. Team Homepage

Pattern Overview	
ID	V-5
Name	TEAM HOMEPAGE
Alias	
Summary	To make it transparent for everybody inside the organization which teams are currently working on what, each team has to maintain a Team Homepage in the internal Wiki.

Example
The case organization has many thousands of employees. To ensure one can find a contact person for a project and get an overview who is working on what, the development teams maintain Team Homepages.

Context
In companies with many teams one cannot know each team and who the members of certain teams are. Teams and members should be findable, to be able to contact them when required.

Problem
C-34: *How to ensure the reuse of enterprise assets?*
C-83: *How to deal with lacking knowledge of another team's activities?*
C-99: *How to deal with discovery of redundancies?*

Forces

- It is often hard to find responsible persons for other projects in case of dependencies.
- Due to the large size of the organization, it is hard to know which projects are existing and which teams are working on them.

Solution
Each team should create and maintain a Team Homepage in the corporate intranet or wiki. The mandatory and optional fields that have to be maintained on the Team Homepage are shown in Table 4.7. The Team Homepage contains information about which product the team is currently working on, who is part of the team, which roles are present, and links to other pages that the team is maintaining. Additionally, the Team Homepage should contain a way of contacting the team. Rows with <...> are optional fields that do not have to be present in every instance of the Team Homepage.

Product	*Name of Product*
Product Owner	*Name of PO*
Scrum Master	*Name of Scrum Master*
Software Manager	*Name of Software Manager*
Developers	*List of all the Developers in the team*
QA	*Name or list of persons responsible for quality*
Technical Writer	*Name of Technical Writer*
Email	*The email to contact the team (manager's email or mailing list of team members*
<Headcount Planning>	*The organization unit where the headcount planning for this team is allocated*
<Headcount Requests>	*The organization unit where to send headcount requests*
<Instant Messaging>	*The address to reach the team at the used instant messaging service*
<Architect>	*Name of Architect*
<Subject Matter Expert>	*Name of Expert for the Project*
<Business Service Owner>	*Name of Business Service Owner*
<Release Manager>	*Name of Release Manager*
<Useful Links>	*List of other useful links (e.g., the code repository, or online agile board of the team)*
<Location>	*The location where the team is sitting*

Table 4.7.: The information fields that have to be maintained on the Team Homepage. Optional fields are marked with <...>.

Variants
The Team Homepage can be adjusted as needed. The Table 4.7 highlights which fields have to be maintained on the Team Homepage to ensure a certain level of information and which fields can be varied. Individual information can be added by the team (e.g., calendar).

Consequences
Benefits:

- Everybody can see which teams are working on which projects.
- Contact persons can be found easily.

Liabilities:

- Additional effort for the team to maintain the page.

Data Collection
Each team maintains the content of its Team Homepage itself. The information should be updated as soon as possible after changes. The Figure 4.7 shows the data model of the Team Homepage.

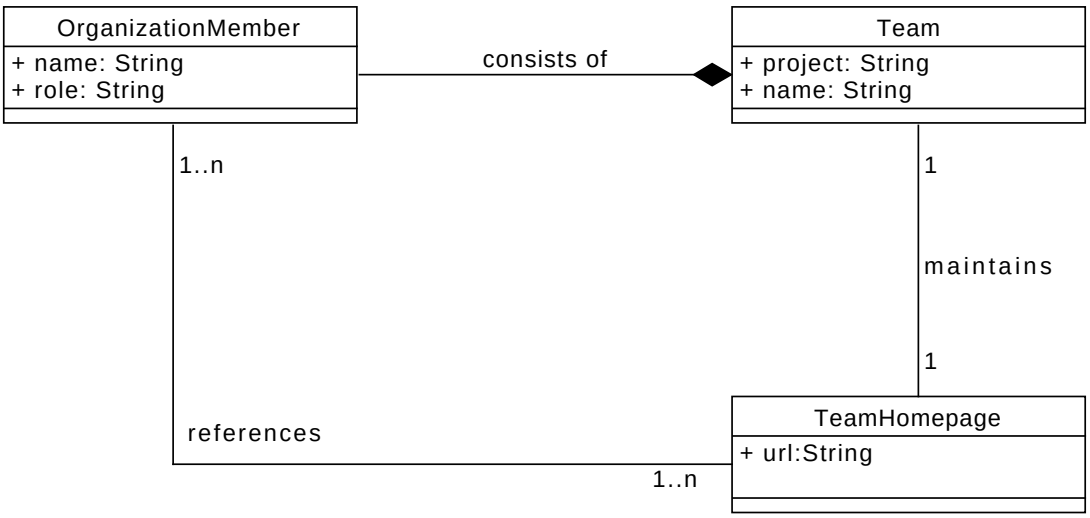


Figure 4.7.: Data Model of the Team Homepage

4.4.8. Background on Team Homepage Practice

The Team Homepage viewpoint practice was used by all observed teams at the organization. While the pages of teams Team4 and Team5 contained only the information fields in the practice's documentation sheet, those of the teams Team2 and Team6 contained additional information such as team calendars, and descriptions of the project requirements and target audience.

4.4.9. Demo Driven Development

Pattern Overview	
ID	AP-2
Name	DEMO DRIVEN DEVELOPMENT
Alias	
Summary	Development that is driven by central marketing demos often does not incorporate a solid architecture and system integration, because the successful marketing demo is valued higher than solid development.

Example
Interviewee PO3 mentioned that the initial development phase of their project was highly driven by a marketing demo, which a board member had already scheduled before development start. The team worked towards this demo deadline, instead of taking enough time to create a solid project architecture and define how the system should integrate with other systems.

Context
The project is in an early development stage. A marketing demo is taken as a development goal or deadline.

Problem
C-8: *How to ensure that non-functional requirements are considered by the Development Team?*
C-117: *How to deal with demo driven development?*

Forces

- Demos are often scheduled by persons outside of the team.
- Demos are important for marketing and sales, however deadlines coming with a scheduled demo increase the pressure on the development team to deliver marketable features instead of a solid product.

General Form
Marketing demos are set up by board members for sales purposes. If these marketing demos are scheduled before the scope of the project is actually defined, this is an anti-pattern. It adds a deadline to the development team before the project could even start up correctly and before a thorough architecture could be defined.

Consequences
Liabilities:

- Team is put under pressure.

- Development work is oriented on what is needed for the demo, not on what is most important for the project.
- Important work such as architecture, documentation, and security is neglected.

Revised Solution
Interviewees PO3 and SA2 both described that a way to revise the solution is to postpone marketing demos until the project is up and running properly.

4.4.10. Background on Demo Driven Development

The Demo Driven Development was described as a concern (**C-117**) and as an anti-pattern candidate (**AP-2**). The concern from team level perspective was how to deal with marketing demos that have been scheduled too early by higher-level management. The anti-pattern candidate from development perspective was that the development focused too much on demo-oriented features instead of thorough solution design. Interviewee PO3 described that the development of their new product was driven by a marketing demo from the start. This hindered the teams to focus on solution design and architecture in the early stage of the project. Instead, it forced them to focus on demo related features. PO3 also drew a strict line between marketing demos and development demos. The latter are, in contrast to marketing demos, useful for development and gathering feedback. A similar situation was described by interviewee M1 and observed in Team2. The case organization held an in-house exhibition during the time of this study. As the exhibition was approaching, Team2 was struggling to meet the deadline. They complained about missing time for sophisticated development and testing. Interviewee M1 named this situation 'Event Driven Development'. Besides this, participant SA2 mentioned in the interview that documentation has been neglected due to this Demo Driven Development. As a result, SA2 recognized an increasing division of knowledge.

4.5. Coordination Mode Analysis of Identified Practices

To get a better overview about when the presented pattern candidates where applied at the case organization, Figure 4.8 shows a mapping of the 'Communication & Coordination' pattern candidates into a coordination mode matrix. The figure shows on the y-axis to which coordination mode, according to [55, 77], each pattern belongs. And on the x-axis it shows how frequent the pattern is applied at the case organization. The Scrum meetings **CO-19**: DAILY STANDUP MEETING, **CO-20**: SPRINT RETROSPECTIVE MEETING, **CO-21**: SPRINT REVIEW MEETING, and **CO-22**: SPRINT PLANNING MEETING were used by all studied teams at the case organization. The documentation in the Appendix C regarding these meetings contains only the most important information, as they are widely known practices and are described thoroughly in the Scrum specification. Further, the following other important coordination mechanisms are also shown in the matrix:

- The **CO-15**: OPEN WORK AREA of the organization's office in Munich contributed considerably to the communication of employees. Interviewee SM1 pointed out that the work space reduced the need of scheduled meetings for him as it is easier to walk over to colleagues to resolve small issues. The open work area is also linked to the ad hoc communication.
- The organization was using a **CO-16**: INSTANT MESSAGING service for asynchronous communication. Each team maintained its own group in the messaging service and general topic groups (e.g., product related groups) were maintained by administrators of the organization.
- Additionally to the instant messaging the organization also maintained a **CO-17**: ONLINE FORUM for questions and answers. This online forum could be used by employees to discuss a variety of topics (such as development work or organizational topics), coordinate workshops, share presentation slides, and more.
- A lot of work coordination at the observed teams was done via **CO-18**: AD HOC COMMUNICATION. People would just talk to each other to clarify things when they met on the floor or in the coffee corner. Unscheduled meetings were held on a daily basis between two or more persons.
- Each of the observed teams used an **CO-23**: ISSUE TRACKING TOOL to document and assign tasks and issues. While teams were using tools from different vendors, the most important aspect was that this tool was accessible for all Scrum team members. It helped people to get an overview of who was working on which tasks and how the team was progressing.

Besides those mechanisms, the case organization also heavily relied on the **M-9**: PAIR PROGRAMMING practice. Interviewees from all three product areas mentioned pair programming as a useful tool of regular use. Especially for knowledge sharing and providing

feature specific help across different locations and teams. The pair programming technique was used as a direct person to person coordination mechanism. This was enabled by video conference and screen sharing tools used at the organization.

The coordination mode matrix in Figure 4.8 highlights that the studied teams at the case organization largely relied on group mode coordination mechanisms. Besides that, the unscheduled ad hoc communication mechanisms (Open Work Area, Personal Ad Hoc Communication via Instant Messaging, and the Online Forum) throughout the three coordination modes were important for the coordination of work. These findings are in line with those of the study on group mode coordination in large-scale agile software development conducted by Dingsøyr et al. [28]. Compared to the determinants of the coordination modes [77], this observation of significant focus on group mode coordination mechanisms is in line with the *task interdependence* determinant. However, it contradicts the hypothesis regarding the *size of work unit* determinant. According to the hypothesis of Van De Ven et al. [77], increased task interdependence yields a significant increase of group mode coordination. The observations indicate that this relation is true at the case organization, as nine interviewees were concerned with **C-84**: *'How to deal with unknown dependencies between teams?'* and we observed high usage of group mode coordination. Further, Van De Ven et al. [77] also state that increased *size of work unit* yields a decrease in group mode coordination. However, as the case organization's Product A team sizes and headcounts have been growing throughout the course of this study, the teams decided to start implementing the Nexus framework on top of the existing Scrum teams in the following months. This can be categorized as an increase in group mode coordination and therefore does not match this hypothesis.

Regarding the impersonal mode coordination mechanisms it should be noted that due to the size of the organization very many impersonal coordination mechanisms were in place. The ones depicted in the coordination mode matrix in Figure 4.8 are those that were used by the observed teams. Other teams, especially at other office locations, might use a different set of impersonal coordination mechanisms.

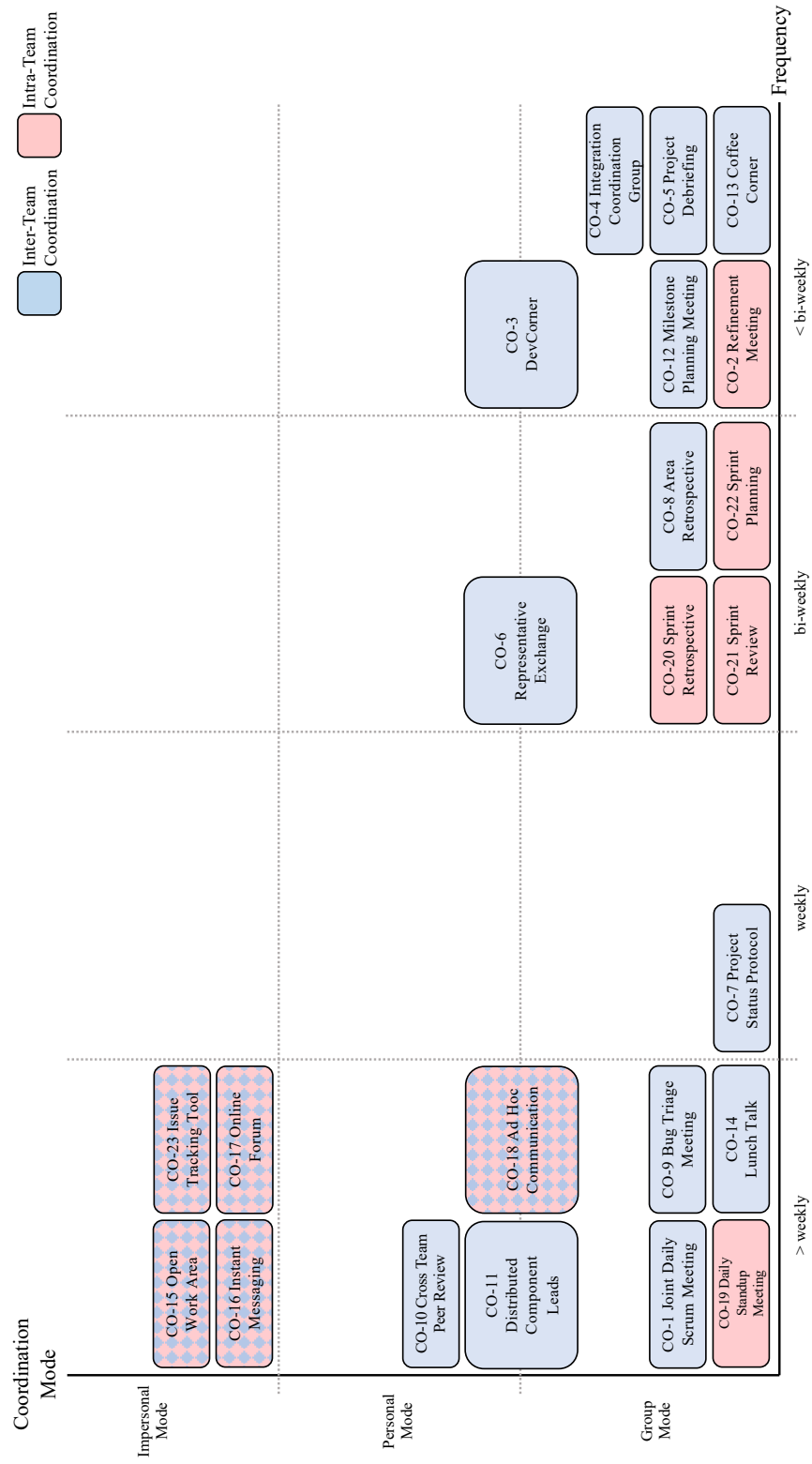


Figure 4.8.: The observed coordination mechanisms at the case organization, categorized by coordination mode

5. Discussion

This chapter reviews the key findings that we learned from the previously presented study at the industry partner. It also discusses quality considerations and limitations of the thesis in Section 5.2.

5.1. Key Findings

Besides the identified concerns and practices, this thesis yields several key findings:

- The pattern language approach of the LSADPL and consequentially this thesis were generally well received by the participants. Most interviewees were familiar with the concept of patterns from the software engineering domain and embraced the translation of this concept to the organizational domain.

- Most of the identified concerns are on team level. This emphasizes the importance of a strong team level agility as a basis for successfully scaling agile methodologies to larger contexts. Team level understanding of agile values is important for large-scale agile development.

- While the focus of this thesis is on coordination and methodology concerns, some of the identified concerns cannot be exclusively assigned to one of these categories. Some concerns can also be categorized into other fields of the LSADPL. Especially the geographic distribution, which has a dedicated concern category in the LSADPL, is connected to many coordination concerns at the case organization.

- Many of the identified concerns cannot be exclusively attributed to one stakeholder-category. On team level, concerns often apply to multiple stakeholder-categories at the same time. Thus, we find that it might be useful for future work to differentiate between concerns that are *relevant* to a stakeholder and concerns that a stakeholder *actively addresses* by taking action.

- The industry partner organization largely relies on group mode coordination. We find that this observation is in line with the assessment of group mode coordination in the literature. Further, inter- and intra-team coordination are balanced at the studied organization.

- The open work area of the industry partner's Munich office significantly facilitates ad hoc communication and personal information exchange between people.

5.2. Limitations

This section gives an overview on quality considerations regarding validity and reliability of the study. In general, this study followed the four principles of data collection of [80].

1. We used multiple sources of evidence in the form of sampling interviewees from different teams in the case organization.
2. We maintained a study database throughout the course of this study. However, it cannot be made publicly available.
3. We established a chain of evidence by coding the data following a defined system of codes.
4. We only added third degree sources to the study database that have been created by internal persons of the case organization.

The *construct validity* describes whether the used operational measures are correct for measuring the studied concepts, and whether data is collected objectively [80]. In this study the construct validity is addressed by using data source triangulation to collect data [80]. We used multiple data sources in the form of interviewees from different teams, documentation maintained by different teams, and observation of meetings of different teams. Further, coding the collected data, as described previously, ensured that a chain of evidence is provided for the findings. However, we cannot make the study database publicly available to support the construct validity.

The *internal validity* describes whether the findings actually portrait the studied concepts authentically [52]. In order to achieve internal validity, we handed out a preparation document to the participants alongside the invitation to the interview. This document described the context and the research project in detail to make sure the interviewee and researcher started the interview from a common ground. Further, the interviewees were presented the list of already identified concerns only after they had identified and described their own concerns and solutions to not influence their answers.

The *external validity* describes whether and how far the findings of the research are generalizable beyond the study and to other contexts [52]. We documented the findings of this study in the form of patterns that address specific concerns in the context of large-scale agile software development. Therefore, the identified concepts may be used by other organizations to evaluate their applicability in other environments. However, the context in which the findings can be applied will most likely stay restricted to large-scale agile development.

The *reliability* describes whether a study has been conducted consistently in process and methods, and whether the study can therefore be reproduced by other researchers in the future [52]. We collected the findings at the case organization over a period of five months

time. However, both the problems and the solutions found in the case organization may evolve over time in the future. Therefore, the findings of this study may not be reproducible by other researchers at the same case organization, let alone at other organizations. As interviewee SM2 stated in the interview: "As usually in agile, maybe in a few months' time we identify something else or maybe come up with a better solution." Hence, the reliability of the findings of this thesis cannot be assured.

Additionally to these important research quality criteria, it should be mentioned here that the interviews as well as the coding of the interviews have been done by one researcher. However, the finished coding was reviewed by a second researcher who is versed in the topic and literature of this thesis.

6. Conclusion

This last chapter summarizes the thesis and gives an outlook for future work.

6.1. Summary

In this section we answer the research questions of the study.
RQ1. What are recurring coordination and methodology concerns in large-scale agile development? To answer this research question, in the semi-structured interviews we asked practitioners to identify their most pressing concerns regarding coordination and methodology in large-scale agile software development. Additionally, we asked them to identify recurring concerns from a list of concerns that we identified in literature and other interviews. In the study we identified 28 new concerns in large-scale agile development that are relevant to three or more interviewees. Further, we found that 34 previously identified concerns are relevant to three or more of the interviewed practitioners. Thus, we consider them being recurring concerns. The findings show that while many of the concerns are on above-team scaling levels the majority of newly identified concerns are found at the team level. This indicates that the frictionless functionality of individual teams is a vital prerequisite for successfully applying agile methodologies at scale. This finding may be attributed to the selected interview participants. Most of them were working inside the actual development teams. As the initial work on the LSADPL focuses mainly on concerns and practices of Agile Coaches and Scrum Masters [35], the findings in this thesis complement the existing ones.
RQ2. What are good practices for addressing recurring coordination and methodology concerns in large-scale agile development?
To answer the second research questions, we asked interviewees to describe solutions they apply in their work to address the identified concerns. Further, we observed the daily work of the teams, as presented in Section 4.2. We identified 38 good practices, divided into 23 coordination practices, 9 methodology practices, and 6 viewpoint practices. Although some of the interviewees mentioned there had been previous tests to apply SAFe, we found that the case organization did not use any framework to apply agile methodologies at large-scale. Rather, the teams themselves decided on how to work and how to collaborate with other teams.
RQ3. Which anti-patterns regarding coordination and methodologies should be avoided in large-scale agile development?
Additionally to identifying good practices, we also asked interviewees whether they ex-

perienced any bad practices when trying to address their concerns. We identified 4 bad practices to answer the third research questions.

6.2. Outlook

We collected all of the presented concerns and practices at the described industry partner. As mentioned earlier, the LSADPL uses the *rule of three* to distinguish between good practices and actual patterns. Therefore, additional qualitative studies at other organizations have to be conducted to find which concerns are recurring across several organizations, and to identify actual patterns among the documented pattern candidates of this thesis. Complementary quantitative studies may be conducted to identify patterns. Further, future observation and study at the industry partner of this thesis could be conducted. Especially with focus on the plans in Product Area A to introduce the Nexus framework for scaling agile development. The transition from the current setup to the Nexus framework and its effects on the concerns and practices may be of interest for future research. This study conducted the first two steps of the PDR approach *observe & conceptualize* and *pattern-based theory building & nexus instantiation*. Hence, future research should build on the results of this thesis to implement the remaining two steps *solution design & application* and *evaluation & learning*. The results of this thesis can be used as foundation for future research to design a solution for an organization that wants to implement scaling agile practices. The application of this solution design and the observation of subsequent deviations at the organization would close the feedback loop between academia and industry that is intended by the PDR.

A. Appendix

A.1. Interview Questionnaire on Identifying Concerns and Good Practices

The following questions were used to conduct the semi-structured interviews for this thesis:

First Part: General Information

1. **Section: Questions about the Participant**
   a) Which role description applies to you?
   b) How many years have you been active in the field of scaled agile software development?

2. **Section: Questions about the Team**
   a) How long has your team been using scaled agile software development?
   b) How many members does your team have?
   c) How many of them are developers?
   d) What other roles are present in your agile team?
   e) Is your team internationally or nationally distributed?
   f) When did you start development of the current project / product?

3. **Section: Questions about the Organization**
   a) How was agile transformation initiated in your company?
   b) What differences do you see in traditional agile development and large-scale agile development?
   c) How do you rate the maturity of your business agility?

Second Part: Identification and Description of Concerns and Practices

1. What are the recurring concerns you face in your role?
2. On what level do these typically occur and how often are you confronted with them?
3. In which category would you classify the concern?
4. Which practices do you apply to address the concern?
5. Which practices should not be applied to address the concern?

Third Part: Identification of Recurring Concerns and Description of Practices

1. Which of the following concern did you face in your role?
2. At what level do they typically occur and how often are you confronted with them?
3. In which category would you classify the concern?
4. Which practices do you apply to address the concern?
5. Which practices should not be applied to address the concerns?

Fourth Part: Discussion

1. Do you have any comments or open points?
2. May we contact you again in the context of the study if necessary?

B. Appendix

B.1. Documentation of Concerns

The tables on the following pages contain all recurring concerns we identified during this study, which were raised by three or more participants. For each concern the attributes for identification, name, description, category, scaling level, and the source are documented.

ID	Name	Description	Category	Scaling Level	Source
C-101	*How to keep the team motivated despite frequent, severe changes in requirements?*	Even though agile values exhort people to embrace change, in the case organization there was still a lot of frustration due to frequent changes in project requirements and scope. Especially in Team1, this concern was relevant for the Scrum Master and Product Owner because in the development work they faced several serious changes in requirements.	Culture & Mindset	Team	SM3
C-102	*How to deal with corporate hierarchies and salary structures?*	Interviewee M1 mentioned he observes that "[...] in traditional corporate structures, which are structured hierarchically, agility reaches its limits." Even though agile teams are structured democratically, in the larger organization "[...] often there are limitations at some point that tell you 'we do this differently here'. You just have to go up far enough, and then there are always some policies or areas where you can't get ahead." Interviewee SM3 also described, that the hierarchical structure in the case organization sometimes intimidates junior people to raise their concerns against senior colleagues.	Methodology	All	SM3, M1
C-105	*How to deal with requirements coming from different sides?*	Interviewee SM1 described, that it is "[...] a problem in big organizations that you have requirements coming from left and right, from any random person at any random time." The problem of this concern is the multitude of stakeholders and communication channels, from which requirements can reach the PO and the development team.	Communication & Coordination	Team	SM1

ID	Name	Description	Category	Scaling Level	Source
C-106	*How to deal with lack of time to work with the Product Owner to organize the backlog?*	This is a concern for Scrum Masters, that describes coordination problems regarding collaboration with the PO. Because the PO often is very busy, and is working between the two sides of business and development, it was expressed by Scrum Masters that they struggle to find time slots to organize the backlog together with the PO. This concern was enforced by the fact that some Scrum Masters were doing this role only part-time.	Communication & Coordination	Team	SM1
C-107	*How to deal with slow reactions of other teams or people in case of dependencies?*	In case of unforeseen dependencies during a sprint, it is a concern for several interviewees that other teams or individuals in the organization often take a long time to reply to inquiries. In the meantime the work in the ongoing sprint is often blocked.	Communication & Coordination	Organization	SM1
C-108	*How to deal with different learning speeds of team members?*	Scrum Master SM1 expressed the concern, that due to the high pressure regarding approaching deadlines, the team was facing issues at dealing with different working and learning speeds of individual developers. It is hard for the Scrum Master to balance the desire of "slower" people to learn new technologies, and the need of the team to finish critical tasks in time and thus assign them to already experienced team members. SM1 found it difficult to ensure a certain balance in this case.	Methodology	Team	SM1

ID	Name	Description	Category	Scaling Level	Source
C-110	*How to deal with increasing complexity of systems based on micro-service architecture?*	Interviewee D1 mentioned the concern, that, due to the distribution of development across many projects and teams, the emerging micro-service landscape is increasingly complex to handle. "Our main focus here is on coordinating the distributed systems", said D1. The micro-service architecture requires additional coordination between teams and integration between components.	Software Architecture	Organization	D1
C-111	*How to keep the team focused on the larger context and project goals?*	Interviewee D1 described situations where the development team Team4 had trouble focusing on the larger goal of their project. The team was very busy doing patches and security related tasks, and the usage of the Kanban methodology in this situation led to them thinking only in 'tickets'. The switch to Scrum improved on this, but did not resolve the concern completely.	Methodology	Team	D1
C-113	*How to ensure acceptance of Product Owner and Scrum Master by the team?*	The manager M2 described that in the past he faced issues with the acceptance of the Product Owner and Scrum Master by the development team. The reason for this was that, though both of them were sophisticated agile practitioners, their missing technical expertise in the area of the project led to disrespectful behaviour by members of the development team.	Methodology	Team	M2

ID	Name	Description	Category	Scaling Level	Source
C-115	*How to take decisions in multiple-team setups?*	In multiple-team setups, there has to be one person that can take decisions in deadlock situations. Interviewee M2 described this concern. M2 added, that neither one of the POs nor the Product Manager are the right person to take such decisions. The concern of M2 was to whether make those decisions personally, or to appoint someone to be able to take decisions that affect multiple teams.	Methodology	Program	M2
C-116	*How to deal with an existing development team before requirements are existing?*	In large organizations, teams often already exist before their project is actually started. This can be due to a previous project of the team being finished, or because staffing has been started before the project is set up. In case of Team1 the latter was the case. The team was created and set up, before the project they were supposed to work on was completely defined. The PO3 mentioned the concern, that this lead to incomplete requirements and changing scope of the project. Ultimately, the initial project of Team1 was even merged into a much bigger project, because the initial project definition was not sophisticated enough.	Communication & Coordination	Team	PO3

ID	Name	Description	Category	Scaling Level	Source
C-117	*How to deal with demo driven development?*	This concern was raised by interviewee PO3. He mentioned that due to a demo that has been scheduled in a very early stage of the project of Team1, the development focused only on relevant topics for this demo. This led to a neglection of important project-specific topics, such as a thorough software architecture. The demo was scheduled by a board member, so the team could not help themselves and had to develop demo driven.	Communication & Coordination	Program	PO3
C-119	*How to deal with issues that interrupt the sprint?*	This concern describes the problem of handling tasks, that are coming in during a running sprint, and require an action by the development team. Interviewee PO1 said: "One of the main concerns with everything in general development is unexpected tasks or things that were not planned for the sprint." Even though in Scrum the Scrum Master should shield the team from outside influence during a sprint, this cannot always be achieved. E.g., for tasks that require immediate action of a technical person, like a developer.	Methodology	Team	PO1

ID	Name	Description	Category	Scaling Level	Source
C-120	*How to coordinate work across multiple time zones?*	Interviewee SM2 mentioned that "[...] the times and working across time zones is the biggest challenge in working in this distributed team." The main reason for working across multiple time zones being a concern is the short overlap times between the time zones to coordinate all the work. In the case organization, the working time overlap between the teams in Montreal and those in Munich was especially short, with only two hours each day. Further, interviewee AC1 added to this concern that this "[...] has an impact on the work live balance". It forces teams "[...] to meet in strange hours that are not really typical for their time zone."	Communication & Coordination	Team, Program	SM2, AC1, D4
C-121	*How to deal with not being able to physically sit together in distributed teams?*	Teams that are co-located can sit together physically in front of their whiteboards to have meetings such as the Daily Scrum and Sprint Plannings. They can concentrate on the work they are doing. In contrast, distributed teams cannot sit together. They have to rely on technology to communicate and replicate the board they are using in the meeting. Their way of holding meetings is restricted by the possibilities of the technology. This concern is also linked to **C-138**: *How to deal with distractions in online meetings?*.	Communication & Coordination	Team	SM2

ID	Name	Description	Category	Scaling Level	Source
C-122	*How to deal with unexpected dependencies?*	This concern describes situations, where teams discover unexpected or unforeseeable dependencies during a running sprint. In a particular case described by interviewee SM2, a developer was absent for a period of time. Other team members continued their work and reached a point, where they could not progress anymore without the task of the absent person being done. The concern can appear both inside teams and between teams.	Communication & Coordination	All	SM2
C-124	*How to implement scaled agile methodologies?*	The participant AC1 mentioned being concerned with actually applying agile values and methodologies in larger scale. Especially, the continuous improvement of work processes and running retrospectives on larger scale is a concern to AC1. Inter-team coordination and communication processes on higher levels are often not as straight forward as on team level, and are therefore hard to grasp and improve by retrospectives.	Methodology	All	AC1
C-125	*How to plan work and track progress?*	While agile frameworks provide methods to plan work in individual teams, they do not provide guidance regarding multiple teams. In the case organization, the concern was raised on how to actually plan work across multiple teams, and how to then check whether the project is progressing in the right direction.	Methodology	All	AC1

ID	Name	Description	Category	Scaling Level	Source
C-127	*How to meet release dates?*	In the case organization projects are often started before their purpose and requirements have been finally decided. This makes it hard to actually define and meet deadlines for the project. This is due to the reason, that the case organization is mostly developing standard software. There is no direct customer driving or pressurizing the development of the product. This concern is loosely linked to **C-116**: *How to deal with an existing development team before requirements are existing?*.	Methodology	Program	M1
C-128	*How to introduce agile practices sustainable?*	Development Managers expressed that while the introduction of agile methodologies was not that big of a problem in their product areas, ensuring that they are adopted sustainable and are applied consistently is a concern to them. They observed that people tend to fall back into old behavior over time. It is a concern to them, to ensure correct application of agile methods over a longer period of time.	Methodology	Program	M1
C-129	*How to avoid wasting time on technical exploration?*	Interviewee M1 faced the concern, that in the case organization time was often wasted on unnecessary technical explorations, like proof of concepts and spikes. One reason for this was the missing clear definition of the Software Architect role, and how this role interacts with the development team.	Methodology	Team	M1

ID	Name	Description	Category	Scaling Level	Source
C-132	*How to increase project visibility in the organization?*	Because the case organization is large and distributed, not everyone can be aware of which projects and products are currently being developed. Interviewee M1 was struggling with incorporating internal marketing effort, in order to make other members of the organization aware of the product they were developing.	Communication & Coordination	Program	M1
C-133	*How to deal with increased demand of status updates in agile methods?*	Interviewee SA1 mentioned that, especially in his architecture work, the Daily Scrum meeting is pressuring people to come up with artificial status updates. In particular, if the work takes longer than expected. This reduces the value of these meetings and is also linked to **C-134**: *How to deal with inefficient coordination meetings?*.	Methodology	Team	SA1
C-134	*How to deal with inefficient coordination meetings?*	This concern was raised by interviewee SA1: "Like I said, I strongly believe that most of the meetings are inefficient." This is on the one hand due to **C-133**: *How to deal with increased demand of status updates in agile methods?*, and on the other hand because of **C-138**: *How to deal with distractions in online meetings?*. Most of the meetings are not relevant to all of their participants. Thus, people get distracted and the value of the meetings decreases for all attendees.	Communication & Coordination	Team	SA1

ID	Name	Description	Category	Scaling Level	Source
C-135	*How to align teams from independent projects to integrate their products?*	The case organization did acquire over 25 other companies during the past ten years. This acquisitions come with additional efforts of integrating their products with the existing product line-up of the case organization. Specifically, Team1 had to integrate the product of a company, that was acquired in 2017, into the project they were working on during the time of this study.	Communication & Coordination	Enterprise	*Observation* and M2
C-136	*How to avoid building up technical debt due to fast iteration?*	This concern was described by participants from the teams Team2 and Team4. Because of pressure from the stakeholders and too short iteration cycles, both teams built up technical debt in the past. It cost them considerable time and effort to fix it afterwards.	Methodology	Team	D1
C-137	*How to balance shielding of the developers and giving them enough project context?*	This is a PO concern described by PO2 and PO3. It is the responsibility of the PO to balance the amount of information that passes from the business side to the development side, and vice versa. It is further also the responsibility of the PO to recognize which concerns from which stakeholders are actually relevant to the project. This is linked to **C-105**: *How to deal with requirements coming from different sides?*. However, also interviewee D1 raised this concern. For developers, the missing information that the PO "filtered" out can often lead to a loss of context. It is therefore also a concern for developers to ensure that they get enough contextual information for their work.	Methodology	Team	D1

ID	Name	Description	Category	Scaling Level	Source
C-138	*How to deal with distractions in online meetings?*	Additionally to **C-134**: *How to deal with inefficient coordination meetings?*, people tend to get distracted in online meetings even more than in face-to-face meetings. Interviewee SM2 mentioned that "[...] you can feel that they [do] not really completely participate in the meeting, because you don't have this under control, right." Interviewee PO1 also added a technical dimension to it: "[...] if you do that on a Skype-Call, like, it is normal that people get disconnected."	Communication & Coordination	Team	D4

B.2. Occurrences of Duplicate Concerns

The following table contains the collected data on concerns that we decided to remove or merge with other concerns. This is intended to give additional information about the data we collected.

ID	Name	Category	#	Percentage
C-103	How to deal with frustration by change despite using agile methods?	Culture & Mindset	9	60%
C-104	How to deal with highly ambiguous tasks in agile methods?	Project Management	5	33%
C-109	How to ensure that quality requirements are prioritized in development work?	Software Architecture	7	47%
C-112	How to avoid developing solutions multiple times in the organization?	Enterprise Architecture	11	73%
C-114	How to deal with urgent bugfix requests?	Methodology	4	27%
C-118	How to deal with resistance to introduction of agile methods?	Culture & Mindset	5	33%
C-131	How to deal with political decisions?	Culture & Mindset	7	47%
C-123	How to deal with sub grouping due to geographical distribution?	Geographical Distribution	7	47%
C-126	How to deal with lack of social binding between teams due to geographical distribution?	Geographical Distribution	8	53%
C-130	How to deal with people that stick with their way of doing things?	Culture & Mindset	7	47%

Table B.1.: List of occurrences of removed or merged duplicate concerns

C. Appendix

C.1. Documentation of Good Coordination Practices

C.1.1. Joint Daily Scrum Meeting

Pattern Overview	
ID	CO-1
Name	JOINT DAILY SCRUM MEETING
Alias	
Summary	Teams that previously worked on independent projects want to integrate their products. To do so, they are co-located for a limited period of time and hold a Joint Daily Scrum meeting each day.

Example
Company A and company B are two different companies that have been acquired by the case organization. They are now intending to integrate two of their products and align APIs. To get to know the other teams and to establish a common understanding of each other's projects, a team of company A is working in the same location as the project teams of company B for a limited period of time. They hold a Joint Daily Scrum Meeting every morning to get to know each others workflows and projects.

Context
Teams from independent projects have to work together, or independently developed products have to be integrated with each other.

Problem
C-1: *How to coordinate multiple agile teams that work on the same product?*
C-135: *How to align teams from independent projects to integrate their products?*
C-141: *How to coordinate multi-vendor teams?*

Forces

- Due to previous independence, the teams do not know each other and the other project respectively. This can lead to 'not invented here' thinking and communication problems.
- Unless the people get to know each other personally, there will be no active knowl-

edge sharing and coordination between the teams.

Solution
Co-locate the teams at one office for a limited period of time, e.g., one month. Both teams are skipping their regular Daily Scrum Meetings and should have a Joint Daily Scrum Meeting instead. This meeting is longer than the normal single-team Daily Scrum Meeting. By doing this, teams are "forced" to get informed about the progress and impediments of the other team, respectively. This period of Joint Daily Scrum Meetings should take place right after the Kick-Off of the integration endeavor.

Variants
The period of time of the Joint Daily Scrum Meetings can be varied as needed.

Consequences
Benefits:

- Teams get to know each other personally.
- Teams learn about the work processes of each other.
- Questions can be asked directly to the responsible person from the other product team.

Liabilities:

- High expenditure for travel and accommodation.
- Enough office space required to relocate an entire team temporarily.

C.1.2. Refinement Meeting

Pattern Overview	
ID	CO-2
Name	REFINEMENT MEETING
Alias	
Summary	To avoid communication gaps between different stakeholders that are working with the project requirements, hold an additional "Refinement Meeting" after the Product Backlog has been created and before the first Sprint is planned.

Example

Context
The stakeholders and the PO have finished the requirements elicitation. The development of the project is ready to start.

Problem
C-44: *How to deal with communication gaps with stakeholders?*

Forces

- The PO knows what the stakeholders told him, but the rest of the team only gets "second hand" information through the PO.
- It is hard to establish a common vision of the overall requirements before development start.

Solution
Introduce an additional Refinement Meeting after the Product Backlog has been created and before the first Sprint is planned. In this meeting, the PO and the Software Architect will explain and discuss the requirements with the whole team. The team discusses, refines, and confirms each requirement.

Consequences
Benefits:

- The requirements are understood by the whole development team.
- A common understanding of the requirements is established between PO, architect, and development team.

Liabilities:

- Increases the number of meetings that have to be done before the actual development work can start.

- Misunderstandings between the PO and the stakeholders are not discovered or resolved.

Other Standards

- Builds on the product backlog refinement of Scrum [67]

C.1.3. DevCorner

Pattern Overview	
ID	CO-3
Name	DEVCORNER
Alias	
Summary	DevCorner is an optional meeting, addressing problems of distributed knowledge and expertise in large-scale agile development. In the DevCorner developers with required expertise share their knowledge with others that need information.

Example
Developer John, sitting in the Munich office with Team6, has to utilize a component that has been developed last sprint by another developer, Jane, who sits with Team8 at the Montreal office. John requests information about the component via the DevCorner channel. Jane sees this request and provides the needed information at the next DevCorner meeting. A colleague of John is interested as well, and joins the DevCorner after he sees the request by John.

Context
Distributed teams work together on a project or have to utilize each others' products.

Problem
C-1: *How to coordinate multiple agile teams that work on the same product?*
C-100: *How to deal with division of knowledge within and between teams?*
C-135: *How to align teams from independent projects to integrate their products?*

Forces

- Knowledge is not shared automatically between different locations or teams.
- If knowledge is shared, this often happens in a person to person way, which leads to the need of sharing knowledge repeatedly with different people.

Solution
Schedule an optional project-wide, preferably virtual meeting across all locations and for every development team member. Establish a dedicated communication channel (e.g., a Slack channel, Confluence page) where developers can request to be informed about topics they need. Other developers that have the required knowledge can provide it to everybody that is interested during the DevCorner meeting. The virtual DevCorner can be attended by developers from all locations.

Variants

The channel through which DevCorner sessions can be requested can be varied according to organization and project needs.

Consequences
Benefits:

- Avoids that the knowledgeable developer or person has to answer the same questions over and over again.
- Establishes a straightforward way to request and share needed information.
- The virtual meeting is location independent and can be joined by everybody who is interested.

Liabilities:

- Success depends on willingness and self-initiative to share information that is requested via the channel.
- A time slot has to be blocked and should not be used for other events, to give everybody the opportunity to participate in the DevCorner.

C.1.4. Integration Coordination Group

Pattern Overview	
ID	CO-4
Name	INTEGRATION COORDINATION GROUP
Alias	
Summary	To coordinate the integration of teams that have previously worked on independent projects, an Integration Coordination Group can be set up by the organization.

Example
A large software company has acquired two other companies, Company A and Company B, in recent years. The products of those two companies should now be integrated and published together in a release. To achieve this, the board sets up an Integration Coordination Group that includes Development Managers, Product Owners, and Software Architects of both companies and a Project Manager.

Context
Two products / projects should be integrated or work together in the future. The teams did not work together in the past and are not familiar (in detail) with the product and processes of the other team, respectively.

Problem
C-1: *How to coordinate multiple agile teams that work on the same product?*
C-135: *How to align teams from independent projects to integrate their products?*
C-141: *How to coordinate multi-vendor teams?*

Forces

- Integration projects are highly ambiguous. Both sides do not have detailed knowledge about the other side, respectively.
- During the integration process complicated decisions have to be made in line with the organization's strategy.

Solution
To achieve successful integration, set up an Integration Coordination Group that includes at least a Development Manager, POs, Software Architects from both sides, and a dedicated Project Manager. They create a high-level solution architecture (not a technical architecture), and define how the integration can be achieved on functional level. The Integration Coordination Group then creates one team of about 10 people for each product / project that is part of the integration. Those teams then identify dependencies on development level between each other —- with the help of the POs and architects — to realize the

defined solution architecture.

Consequences
Benefits:

- The Integration Coordination Group can solve the first level of ambiguity.
- The members of the group are aware of the organizations strategy and have specific knowledge how to take top level decisions in line with the companies' constraints.

C.1.5. Project Debriefing

Pattern Overview	
ID	CO-5
Name	PROJECT DEBRIEFING
Alias	
Summary	The Project Debriefing is a meeting, where the management and other project stakeholders personally explain their reasons to the development teams for taking an impactful decision. This helps to avoid misunderstandings and reduces the frustration of the development teams.

Example
At the case organization, the Head of Technology holds Project Debriefings with the development teams after project scope changes, abortion of projects etc. In these Project Debriefings the Head of Technology and other stakeholders explain their reasoning for their decision and how this is embeds in the strategy of the larger organization.

Context
After the project scope changes (drastically), or after a project has been killed exceptionally. The team shows frustration due to communication reasons or drastic changes.

Problem
C-7: *How to deal with doubts in people about changes?*
C-81: *How to deal with missing communication of decommitment?*

Forces

- Due to business strategy and the context of the organization, the focus points can change. Thus, projects can be killed before finishing with a viable product.
- People tend to get frustrated if their work is "thrown into trash" and they do not know why.

Solution
It is important that the management and other important stakeholders really explain their reasoning for certain decision they have made. The team should understand why a project changed so drastically, or why a project is no longer relevant to the organization as a whole. Therefore, the management and other project stakeholders should hold personal (preferably with physical presence) Project Debriefings and explain their views to the development team. This gives the development team members the possibility to ask questions, and to understand why the management has made this decision. Miscommunication is avoided by meeting personally instead of via third parties. Frustration is reduced by clari-

fying the questions directly with the team. The Project Debriefing should take place in the near time after the decision has been made.

Variants
The way the Project Debriefing is hold, is completely individual.

Consequences
Benefits:

- Reduces the danger of "silent mail" misunderstandings.
- Reduces frustration of the development team because they understand the reasoning behind the decision.

Liabilities:

- Often, it's very hard to find a time slot where all stakeholders are available to attend this Debriefing personally.
- The Debriefing is necessary shortly after the decision to really be effective.

C.1.6. Representative Exchange

The practice **CO-6**: REPRESENTATIVE EXCHANGE was already presented in the findings in Section 4.4.5.

C.1.7. Project Status Protocol

Pattern Overview	
ID	CO-7
Name	PROJECT STATUS PROTOCOL
Alias	
Summary	To efficiently use the short overlap times of working hours between different time zones, a weekly Project Status Meeting is scheduled where the Product Owners and Scrum Masters sync what their teams will be doing in the upcoming period. This speeds up daily communication between time zones in the following days.

Example
At the case organizations Product E it is hard to keep track of who is working on what, due to the large amount of teams. Because of the sort time zone overlaps, communication has to be as efficient and direct as possible. The Product Owners and Scrum Masters of the teams meet in a weekly Project Status Meeting to discuss their teams' status and what they will do in the upcoming week. This helps teams to understand whom to address in case of issues or dependencies and reduces the amount of necessary introduction about the work when communicating.

Context
Multiple teams are collaborating on a project. The teams are distributed across multiple time zones. The overlap of working hours is very limited because of the time zones.

Problem
C-83: *How to deal with lacking knowledge of another team's activities?*
C-120: *How to coordinate work across multiple time zones?*

Forces

- In multiple-team projects it is hard to keep track of what others are working on.
- When communicating with others, often detailed introduction about ones current work is needed. This takes a lot of valuable time.

Solution
The Project Status Meeting is a meeting between Product Owners and Scrum Masters of the participating development teams of the project. They agree on a weekly or bi-weekly meeting schedule. This meeting's purpose is to exchange information about the status of the work of each team, and what each team is planning to do in the upcoming period until the next Project Status Meeting. This helps the participants to get an overview, what is currently happening in the project, and who is working on what. Therefore, through-

out the working period less time has to be wasted on exchanging information about what the teams are doing. The short available time for communication can be used more effectively for resolving issues and hand-overs between the time zones. This meeting can also be used for planning of exceptional periods. For example for setting up a week with extended working hours overlap by arranging one team to come to office earlier and another team to stay a bit longer.
If a team member feels the need to get more specific knowledge about a certain topic or wants to share knowledge with others, he or she states so in the Daily. A **M-9**: PAIR PROGRAMMING is set up in the remaining overlap time. During this pairing the two or more people exchange the information that is necessary to continue the work in the next time zone.

Variants
It is important to agree on a project wide communication protocol to exchange status updates regularly. Thus, the Project Status Meeting can be scheduled according to the project needs either weekly or bi-weekly. Also, a clear mechanism should be defined, how information is exchanged during time zone overlaps (e.g., **M-9**: PAIR PROGRAMMING). This mechanism can be varied.

Consequences
Benefits:

- Sets a common ground for all teams for the upcoming period.
- Makes sure the teams are aligned and work is distributed evenly in a non-blocking way.
- Extensive knowledge transfer is provided between the different locations by a defined mechanism.

Liabilities:

- It is the Product Owner's and Scrum Master's responsibility to communicate the results of the Project Status Meeting to the teams. The effectiveness of the Project Status Meeting, and thus of the whole Project Status Protocol, depends on how well they transfer the results.
- During the pairing two or more people are working on the same task.

See Also

- **M-9**: PAIR PROGRAMMING

C.1.8. Area Retrospective

Pattern Overview	
ID	CO-8
Name	AREA RETROSPECTIVE
Alias	
Summary	The Area Retrospective is a retrospective that takes place in multi-team setups after the individual teams held their own retrospectives. Representatives from the teams participate in the Area Retrospective. They discuss problems on above-team levels and how the teams work together.

Example
At the case organization AC1 is conducting Area Retrospectives to discuss challenges that individual teams cannot solve. This helps to reflect on how the teams collaborate and how to improve this collaboration. AC1 uses these Area Retrospectives to help project teams to improve their collaboration, and also to learn about which challenges regarding agile methodologies are existing across the organization.

Context
Multiple teams working together on one product. Teams want to improve their collaboration. Teams want to reflect on their processes and possibilities for improvement.

Problem
C-1: *How to coordinate multiple agile teams that work on the same product?*
C-124: *How to implement scaled agile methodologies?*

Forces

- Retrospectives are often only conducted inside individual teams.
- Collaboration between teams needs more coordination and processes than on team level. These processes can hardly be improved using team level retrospectives.

Solution
The Area Retrospective takes place after each team held their individual team retrospective. Representatives and Scrum Masters from the teams take part in the Area Retrospective, as well as an Agile Consultant / Agile Coach. In the Area Retrospective above-team level problems are discussed, that were raised in the team retrospectives. In the Area Retrospectives the participants reflect on how the teams collaborate and which processes work or do not work. They decide how the work and collaboration can be improved and take the results back to the teams. The Area Retrospective can be run just like a normal retrospective, but with focus on above-team level processes.

Variants
The way the Area Retrospective is done can be varied. E.g., the **V-4**: SAILBOAT RETROSPECTIVE can be used.

Consequences
Benefits:

- Teams can reflect about what to improve in their work also on higher levels.
- People will be more engaged in the work because they are involved in deciding how they work together.
- Teams can raise challenges to the area.

Liabilities:

- If the outcomes of the retrospectives are not transformed into actions, people will become demotivated.

See Also

- **V-4**: SAILBOAT RETROSPECTIVE

Other Standards

- Large-Scale Scrum (LeSS) [48]
- Nexus [66]

C.1.9. Bug Triage Meeting

Pattern Overview	
ID	CO-9
Name	BUG TRIAGE MEETING
Alias	
Summary	The Bug Triage is a meeting between the Chief Product Owner and the Product Owners, where they discuss newly created issues and bug reports. They judge whether a new bug is actually valid, and assign them to an appropriate team to fix it.

Example
The Product E of the case organization has been released last year. Since then, it scaled up to about 30 live customers and 200 sold instances. To deal with the large number of bugs reported during this after-release scaling period, they set up the Bug Triage Meetings.

Context
Multiple teams working together. In situations after a new product launch / release, when a lot of bugs are coming in that need to be fixed. Often, a certain reaction time to reported issues is obligated by contract.

Problem
C-1: *How to coordinate multiple agile teams that work on the same product?*
C-83: *How to deal with lacking knowledge of another team's activities?*
C-85: *How to deal with increase in geographic dispersion?*
C-95: *How to rapidly deliver a necessary patch?*

Forces

- In special situations, e.g., after a new product launch, often a lot of bugs are reported.
- The incoming issues have to be assigned to the teams in a balanced way and according to competency.
- Ensuring fast reaction times during running sprints is hard.

Solution
Schedule a Bug Triage Meeting to take place two times a week. In the Bug Triage Meeting the Chief Product Owner and the teams' Product Owners discuss newly created issues and reported bugs from customers. The meeting can be hosted by any of the Product Owners, to balance workloads between the Product Owners. They decide whether the reported bugs are valid bugs, whether they have been already addressed elsewhere, or are 'considered a feature'. If a bug is actually valid, the Product Owner group then decides which team should take care of the fix based on competency – which team has the required

knowledge to fix it? – and current workload of the teams. The Bug Triage Meeting is used in scenarios where a lot of bugs are reported from customers.

Variants
Can be combined with the **M-4**: FOLLOW THE SUN practice to ensure quick reaction times. The schedule of the Bug Triage Meeting can be varied according to the project needs and volume of incoming bugs.

Consequences
Benefits:

- Product Owners are informed about current issues and which teams are working on which tasks.
- Incoming bugs can be filtered before development addresses them.
- Bugs are purposely assigned to competent and available teams.
- The reaction time is faster because the reported customer issues are not going through the normal Sprint Planning procedure.

Liabilities:

- Only takes place two times per week. In worst case, a reaction to a reported bug can take a few days.

See Also

- **M-4**: FOLLOW THE SUN

C.1.10. Cross Team Peer Review

Pattern Overview	
ID	CO-10
Name	CROSS TEAM PEER REVIEW
Alias	
Summary	Any work product has to be reviewed by peers in other teams. Not only development work is reviewed, but also requirements, user stories etc. are peer reviewed among the Product Owners. The reviews are assigned to other peers across teams and locations.

Example
At Product E of the case organization any kind of work product has to be reviewed by a suitable peer from another team. This helps the seven teams of the project to keep up to date about what is happening and enforces the principle of dual control.

Context
Multiple teams are working together on a common project / product. Work has to be reviewed by peers to ensure good quality.

Problem
C-32: *How to deal with lacking team cohesion at different locations?*
C-83: *How to deal with lacking knowledge of another team's activities?*
C-100: *How to deal with division of knowledge within and between teams?*

Forces

- People tend to ask their team mates and near colleagues to review their work.
- In geographically distributed projects and teams, often local sub groups emerge that confirm and reinforce their local colleagues' work.
- Teams often assign their reviews inside the same team which prevents other teams from getting insights into the ongoing work.

Solution
The Cross-Team Peer Review is different to the normal review process in software development. It not only applies to development work, but also to other work such as requirements or user stories that are formalized by Product Owners. The work that has been done has to be reviewed by peers from other teams, and preferably even from another location as well. This gives the assigned reviewer the possibility to get insights into what the other teams or colleagues (at other locations) are actually doing.

Variants

The teams should agree on some distribution of the peer reviews. Whether it is across locations, teams, or even both can be varied.

Consequences
Benefits:

- Knowledge about current work is distributed across teams and locations.
- Other teams have an influence on what is actually merged or added as a requirement.
- Nepotism among teams or local sub groups is prevented.

Liabilities:

- This still only involves one other person from a single other team.
- Reviews take place after the actual work has been done, so misunderstandings are only discovered after the work has been done.

C.1.11. Distributed Component Leads

Pattern Overview	
ID	CO-11
Name	DISTRIBUTED COMPONENT LEADS
Alias	
Summary	To ensure that an expert for each feature of the project is available all the time, Component Leads are distributed and replicated across time zones or continents. This is done in such a way, that in every time zone or continent a competent expert is available for each feature during the local working hours.

Example
The Product E of the case organization was piloting the Distributed Component Leads practice during the time of this study. Especially because of the short time zone overlap between the European located teams and the American located teams, they want to have component leads and experts for each feature in both locations.

Context
The teams working together on a project are geographically distributed. Features are distributed among the different teams. Component Leads for the different features are only available during their time zone's working hours.

Problem
C-1: *How to coordinate multiple agile teams that work on the same product?*
C-85: *How to deal with increase in geographic dispersion?*
C-100: *How to deal with division of knowledge within and between teams?*
C-107: *How to deal with slow reactions of other teams or people in case of dependencies?*
C-120: *How to coordinate work across multiple time zones?*

Forces
- Features are developed by teams. Thus, the team that developed it is a centralized expert-team about this feature.
- Teams in other time zones face problems in communicating and access the knowledge of these experts.

Solution
Multiple Component Leads are assigned for the different components and features of the project. The Experts are distributed across the different time zones and continents where the project's development teams are sitting. The distribution is done in such a way, that in every time zone and in every continent, there is always a local Component Lead available

for each feature during the local working hours. This ensures every development team has access to experts for the features of the project. It reduces the waiting times in case of dependencies to other features.

Consequences
Benefits:

- Response times in case of knowledge requests or dependencies to other teams are reduced, because an expert is always available.

Liabilities:

- There may still be one main expert / expert team that has developed the component and has the most deep knowledge about it. This practice stands in contrast to the concept of teams “owning” features.

C.1.12. Milestone Planning

Pattern Overview	
ID	CO-12
Name	MILESTONE PLANNING MEETING
Alias	
Summary	The Milestone Planning Meeting is a meeting between the Chief Product Owner and the teams' Product Owners. In the Milestone Planning the Product Owners discuss what should be achieved in the upcoming two to four sprints.

Example
The product area of AC1 is planning and distributing their work across teams using the Milestone Planning Meeting.

Context
Multiple teams working together on a project. The work has to be planned and distributed to the teams by the Product Owners.

Problem
C-1: *How to coordinate multiple agile teams that work on the same product?*
C-83: *How to deal with lacking knowledge of another team's activities?*
C-125: *How to plan work and track progress?*

Forces

- It is not easy for Product Owner to keep the overview of what is going on in other teams.
- Centralized coordination and planning of all the teams' work is complicated for the Product Owners and the Chief Product Owner.

Solution
The Milestone Planning is a planning event in which the Chief Product Owner and the Product Owners of the individual teams take part. Before the Milestone Planning takes place, the Chief Product Owner defines a "theme" or common goal for the whole project. This theme will be the objective of the upcoming period (usually 2 to 4 sprints). The team Product Owners select possible work for their teams according to the theme that has been defined by the Chief Product Owner. After that, the Product Owners consult with their development teams on whether the selected work is achievable. The team estimates the proposed work packages in story points. Finally, after all Product Owners selected work and the teams estimated it, the Milestone Planning meeting is held by the Chief Product Owner with all the Product Owners. In the Milestone Planning Meeting the Product Own-

ers discuss the planned work and the estimates of the teams. If adjustments are needed, they can be done in the meeting. Workloads might be moved to other teams that do have more capacity.

Variants
This pattern was also described with a longer planning period of a quarter. Thus, the Milestone Planning Meeting can also be conducted quarterly.
The Scrum Masters may also take part in the Milestone Planning to represent the development team and their concerns.

Consequences
Benefits:

- The Product Owners know what is being worked on in all the teams.
- The workload is balanced between the teams.

See also

- **V-1**: ROADMAP
- **V-3**: MILESTONE PLANNING BOARD

C.1.13. Coffee Corner Meeting

Pattern Overview	
ID	CO-13
Name	COFFEE CORNER MEETING
Alias	
Summary	The Coffee Corner is an informal meeting. Top-level management members visit an office of the organization and sit together with interested employees for a cup of coffee.

Example
At the case organization's office in Munich, Coffee Corner sessions are held. They are held in an irregular schedule whenever a board-level or high-level manager is available at the location or visiting.

Context
Direct communication between top-management and employees holds valuable information for both sides.

Problem
C-65: *How to deal with office politics?*
C-142: *How to deal with decisions on higher levels reaching lower levels?*

Forces

- Top-management is often located at the headquarter of an organization.
- Top-management is hardly contactable in large organizations.

Solution
The Coffee Corner is an informal meeting where top-level management members visit an office of the organization and sit together with the employees for a cup of coffee. The managers give an overview about general news in the company (e.g., restructurings, recent events and demos, etc.). The session should be held in an interactive way to ensure bi-directional communication. There should also be a Skype-Meeting going on besides the physical meeting, to enable everybody to join the session.

Variants
The session can be held with or without presentation slides by the manager.
The session can take place in the coffee corner of the office or in any other location.

Consequences
Benefits:

- Information is passed to the employees directly from the high-level management.
- People can ask direct questions to management.

Liabilities:

- Top-Level managers often do not find the time to host Coffee Corner sessions regularly.

C.1.14. Lunch Talk

Pattern Overview	
ID	CO-14
Name	LUNCH TALK
Alias	
Summary	The Lunch Talk is a presentation during the lunch break, where members of the organization present their projects and share information about what is going on in their teams.

Example
Interviewee M1 and one of the teams of Product C are working on a new log analysis solution. He wants to share his knowledge and draw attention of colleagues to this project, because he thinks it might be useful in other projects of the organization. He gives a lunch talk, where he can explain his project to interested colleagues, in a warm atmosphere over some pizza.

Context
People need to be aware of internal projects in order to utilize them. In large organizations employees cannot be aware of all projects that are going on.

Problem
C-83: *How to deal with lacking knowledge of another team's activities?*
C-100: *How to deal with division of knowledge within and between teams?*
C-132: *How to increase project visibility in the organization?*

Forces

- Knowledge about new projects does not automatically spread across the organization.
- Sharing knowledge via wiki pages or newsletters often does not draw attention of colleagues from other projects.

Solution
Set up a Lunch Talk series at the company's different office locations. The events of the series should take place during lunch time. There should be the possibility to have lunch during the presentation. To increase attractiveness, consider offering drinks or food. Project teams or members of project teams can schedule talks in this series, in which they inform the participants about what they are currently developing and how others can utilize this in their projects. The atmosphere should be casual.

Variants

Consider offerings like drinks or food during the lunch talk to increase attractiveness for attendees.

Consequences
Benefits:

- Participants are informed about new projects and how they can utilize them.
- Questions can be pointed directly to the presenter of the project team.

Liabilities:

- More effort than a wiki page or an email newsletter.

C.1.15. Further Coordination Pattern Candidates

ID	Name & Problem	Summary
CO-15	OPEN WORK AREA **C-134**: *How to deal with inefficient coordination meetings?*	The open work area of the organizations' office in Munich contributed considerably to the communication of employees. Interviewee SM1 pointed out that the work space reduced the need of scheduled meetings for him as it is easier to walk over to colleagues to resolve small issues. Therefore, the open work area is also linked to the ad hoc communication.
CO-16	INSTANT MESSAGING **C-121**: *How to deal with not being able to physically sit together in distributed teams?*	The organization was using an instant messaging service for asynchronous personal and topic related communication. Each team maintained its own group in the messaging service and general topic groups (e.g., product related groups) were maintained by administrators of the organization.
CO-17	ONLINE FORUM **C-121**: *How to deal with not being able to physically sit together in distributed teams?*	Additionally to the instant messaging the organization also maintained an online question and answer forum. This online forum could be used by employees to discuss a variety of topics (such as development work or organizational topics), coordinate workshops, share presentation slides and more.
CO-18	AD HOC COMMUNICATION **C-134**: *How to deal with inefficient coordination meetings?*	A lot of communication and work coordination was done ad hoc at the observed teams. People would just talk to each other to clarify things when they met on the floor or in the coffee corner. Unscheduled meetings were held on a daily basis between two or more persons.
CO-19	DAILY STANDUP MEETING **C-111**: *How to keep the team focused on the larger context and project goals?* **C-122**: *How to deal with unexpected dependencies?*	The team conducts a Daily Standup Meeting every morning. It usually takes around 20 minutes. The meeting is used to inform other team members about ones current status and impediments. In case of unexpected dependencies the team consults on how to resolve them. The meeting helps keeping all team members informed about what others are doing at the moment. **Other Standards**: Scrum [65]

ID	Name & Problem	Summary
CO-20	SPRINT RETROSPECTIVE MEETING **C-39**: *How to establish a culture of continuous improvement?*	All observed teams at the case organization used a Sprint Retrospective Meeting to recap on the work processes of the previous sprint. The team identifies what went well, what needs to be improved, and how this improvement can be achieved for the upcoming sprint. **See Also**: **AP-1**: RANTROSPECTIVE, **V-4**: SAILBOAT RETROSPECTIVE **Other Standards**: Scrum [65]
CO-21	SPRINT REVIEW MEETING **C-125**: *How to plan work and track progress?*	The team conducts a Sprint Review Meeting after a sprint has been completed. The created product increment is examined in detail. In larger projects, this meeting is also often used to demo each newly created feature to all the team members. The team revisits what has been achieved, what could not be finished in the past sprint, and adapts the product backlog if needed. **Other Standards**: Scrum [65]
CO-22	SPRINT PLANNING MEETING **C-125**: *How to plan work and track progress?*	The team conducts a Sprint Planning Meeting before a new sprint is started. The team sits together and decides which tasks from the Product Backlog should be pulled to the sprint backlog for the next sprint. The Sprint Planning Meeting is also often used for organizational issues like planning travel between locations. **Other Standards**: Scrum [65]
CO-23	ISSUE TRACKING TOOL **C-125**: *How to plan work and track progress?*	The Product Backlog and the Sprint Backlogs are documented and maintained in a dedicated tool. All Scrum team members have access to the tool. Backlog items can be assigned to one or more person using the tool.

C.2. Documentation of Good Methodology Practices

C.2.1. Mono-Repo

Pattern Overview	
ID	M-1
Name	MONO-REPO
Alias	Single Repository
Summary	To reduce dependencies between multiple teams working on the same product, the entire codebase of the product is kept in one place.

Example
The teams of Product A at the case organization decided to run the newly set up project in mono-repo mode, to facilitate reuse of components and code from the ground up.

Context
Projects usually are separated into components. The components are depending on each other. The components are developed by different / multiple teams. Work has to be somehow separated between teams.

Problem
C-1: *How to coordinate multiple agile teams that work on the same product?*
C-34: *How to ensure the reuse of enterprise assets?*
C-83: *How to deal with lacking knowledge of another team's activities?*
C-122: *How to deal with unexpected dependencies?*

Forces
- Not every team can have full knowledge about how solutions from other teams work and what exactly they have developed.
- The components developed by different teams have to work together in the end. Not every progress step can be tested with all other components.

Solution
Keep all the codebase for all the components for a project in one repository. This reduces the impact of dependencies between components, as interfaces might not have to be mocked anymore. Code that has already been developed in another component can be reused easily. Dependencies, integration errors, or failures can be discovered earlier.

Variants
The structure of the mono-repo should be adjusted to the structure of the project and

teams.

Consequences
Benefits:

- Reduces code redundancies between components.
- Makes code easier to search.
- Helps to discover dependencies earlier.
- Integration of components is easier and can be tested earlier / quicker.
- Errors / wrong development can be discovered earlier.

Liabilities:

- The repository can get very huge, good structuring is needed.

Known Uses
Besides the case organization, this practice is also used at other organizations, e.g., Google [57] and Facebook [34].

C.2.2. Mixed Sprint

Pattern Overview	
ID	M-2
Name	MIXED SPRINT
Alias	
Summary	In situations where a team has to help out another independent team for a short time, the helping team can collect tasks in a separate Epic item. The team then carries out Mixed Sprints for this short time span, during which they work on work-items from their own project, as well as from the other project they are helping out.

Example
Team2 was working on the Product B project when they had to help out on another team's project. The other team had difficulties meeting their deadline. They decided to run mixed sprints for this period of time to avoid additional organizational overhead.

Context
A team needs help from another team on some of their tasks. The time span for which the help is needed is limited from the beginning on.

Problem
C-1: *How to coordinate multiple agile teams that work on the same product?*
C-79: *How to synchronize sprints in the large-scale agile development program?*
C-127: *How to meet release dates?*

Forces

- The helping team has its own workflow and tool setup. Adjusting the setup of the helping team is a lot of overhead work which will be useless after the limited time span.
- Because the help is needed urgently, there might not be enough time to setup a new workflow and toolset for the collaboration.

Solution
The helping team creates a separate Epic in their work items. This separate Epic contains all the tasks that are related to the other team's project that needs the help. The helping team carries out mixed sprints in which some of the developers are assigned to the tasks of the other project. The other developers keep on working on the team's own project.

Variants
Can be combined with a reduction of sprint lengths to reduce the reaction time of the help-

ing team to changing information.

Consequences
Benefits:

- Reduces the time that is needed until the helping team can start contributing to the other project.
- The helping team does not have to adjust to a new workflow and toolset but can continue using the known methods.

Liabilities:

- The project space (Scrum board etc.) of the helping team contains items not related with their actual project.

C.2.3. Assigning Rights

Pattern Overview	
ID	M-3
Name	ASSIGNING RIGHTS
Alias	
Summary	To ensure the acceptance of the persons in the roles of Scrum Master and Product Owner, they are assigned additional rights and responsibilities. They allow them to maintain their position and take decisions.

Example
At the Product Areas A and B the Scrum Masters and Product Owners are assigned the right to setup the Scrum Teams and Methodology on their own. Further, the Scrum Masters can also intervene the Software Manager in personnel questions, e.g., if the Software Manager is moving developers between projects. The Scrum Master and Product Owner are considered leadership roles.

Context
In the Scrum standard the team is designed completely democratically, all members of the team are seen as equal.

Problem
C-113: *How to ensure acceptance of Product Owner and Scrum Master by the team?*

Forces

- Scrum Master and Product Owner have to instruct others, but do not have authority towards the team.
- The Scrum Master has to handle impediments of the team, but has no additional rights than a developer.

Solution
To make sure the Scrum Master and the Product Owner are accepted by the team, they are assigned additional rights and responsibilities by the Manager. Therefore, they can work effectively with the teams and ensure the teams' integrity. These rights can include (but are not limited to):

- Setup the team how they think is right for the project (even disregarding the team's opinion).
- The Scrum Master can intervene the Software Manager in personnel questions.
- The Scrum Master can direct a developer how to do something in cases where no democratic solution is found in the team.

Those additional rights and responsibilities deviate from the Scrum standard. They have to be documented somehow, in order to make them comprehensible.

Variants
The exact additional rights that the Scrum Master and Product Owner are assigned can be varied. Further, the way how these rights are documented can be varied as well.

Consequences
Benefits:

- Scrum Master and Product Owner are clearly defined as leadership roles.
- They have additional 'handles' to deal with special situations and to make decisions.

Liabilities:

- The democratic approach of Scrum is modified.
- Additional hierarchy is added.

C.2.4. Follow the Sun

The practice **M-4**: FOLLOW THE SUN was already presented in the findings in Section 4.4.1.

C.2.5. ShipCaptain

Pattern Overview	
ID	M-5
Name	SHIPCAPTAIN
Alias	
Summary	To have somebody who is able to make global decisions in a multiple-team setup, the 'Ship-Captain' role is introduced. This role is filled by a manager that can make global decisions and override decisions of sub-teams and the Product Owner & Product Manager. The roles purpose is to ensure the program is moving in the right direction.

Example
The case organization has a multi-team setup on Product A. To ensure that the whole program is moving to the right direction and meets the deadlines, they introduced a Ship-Captain.

Context
Multiple-team setups where critical deadlines have to be met.

Problem
C-80: How to deal with a competing concept deadlock?
C-115: How to take decisions in multiple-team setups?
C-127: How to meet release dates?

Forces

- No clear decision taker is defined in multiple-team setups.
- Product Owner, Product Manager, and teams might want to go different directions or take contradictory decisions.

Solution
To ensure that the team is moving in the right direction and will meet deadlines, a new role of a 'Ship-Captain' is established in the multi-team setup. This role is placed at the multi-team level and can take decisions – even if they might be wrong. Its purpose is to ensure the teams are not blocked. The Ship-Captain can also override decisions of the Product Owner and sub-teams if there are reasons for this (like cost, resources etc.). Further, the Ship-Captain can also approve urgent budget requests by the team (to a certain extent) without adhering to the corporate approval procedure. The Ship-Captain is assigned pretty high responsibility and should be filled e.g., by a manager.

Consequences
Benefits:

- Clear responsibilities for steering the program.
- Deadlines are watched and ensured by the Ship-Captain.
- Decisions can be taken very quickly if needed.

Liabilities:

- Product Owner and team decisions might be disrespected.

C.2.6. Sprint Zero

The practice **M-6**: SPRINT ZERO was already presented in the findings in Section 4.4.3.

C.2.7. Dedicated Person to Deal with Annoyments

Pattern Overview	
ID	M-7
Name	DEDICATED PERSON TO DEAL WITH ANNOYMENTS
Alias	
Summary	To reduce the distractions of the developers during an ongoing sprint, a person is dedicated to dealing with unexpected annoyments during each sprint. This role is rotating across the development team members to keep frustration low.

Example
The Team4 has a rotating role that is assigned to another developer each sprint. This developer has to deal with technical issues during the sprint that cannot be handled by the Scrum Master, in order to keep the rest of the team focused on the sprint.

Context
Unexpected issues are coming up during sprints. Organizational topics have to be discussed during a sprint. However, developers should be able to concentrate on the assigned tasks during the Sprint.

Problem
C-95: *How to rapidly deliver a necessary patch?*
C-119: *How to deal with issues that interrupt the sprint?*

Forces
- The team cannot control when the requests are coming in.
- The Scrum Master cannot deal with all the technical requests on his/her own.
- Organizational topics and calls are often necessary to clarify immediately for the next work steps.

Solution
During each sprint, dedicate one specific developer of the team to deal with the incoming requests. This person is assigned the role to handle incoming annoyments, requests, and calls that cannot be handled solely by the Scrum Master. Preferably, this dedicated person is a developer who knows the technical details of the project. The role of the annoyments handler is rotating across the development team members to avoid one member getting frustrated by this job. Otherwise, frustration might occur because the role reduces the person's productivity on the project.

Variants

The rotation scheme of the role can be varied. The important point is to actually have a rotation, to avoid team members getting frustrated by this role.

Consequences
Benefits:

- The development team can really focus on the sprint tasks and does not have to deal with these requests.
- The Scrum Master is assisted by a technically experienced person.

Liabilities:

- The dedicated person does not get to the main job and is less productive during this sprint.

C.2.8. Requirement Separation

Pattern Overview	
ID	M-8
Name	REQUIREMENT SEPARATION
Alias	
Summary	To stay productive despite changing project requirements, the team separates tasks that are very dependent on specific requirements and such tasks that are rather independent from the detailed requirements. Focus is kept on the former in the beginning phase of the project.

Example
The requirements of the initial project of Team1 changed severely multiple times, because upper management had no clear vision of the project in the beginning. The team got demotivated. This was addressed through the Scrum Master and Product Owner, by focusing on tasks that were not deeply dependent on the exact requirements, but rather on the general scope of the project. This helped to avoid "throwing finished work into trash" each time requirements changed.

Context
The customer has no fixed vision of the project. Requirements are not clearly defined by the customer.

Problem
C-7: *How to deal with doubts in people about changes?*
C-101: *How to keep the team motivated despite frequent, severe changes in requirements?*
C-116: *How to deal with an existing development team before requirements are existing?*
C-137: *How to balance shielding of the developers and giving them enough project context?*

Forces
- Teams repeatedly do work for the "trash" because requirements can always change.
- Teams get demotivated because of this and lose trust in the project and in themselves.

Solution
Separate tasks from the task-board into two categories:
- Tasks that are tightly linked with specific requirements.
- Tasks that are (almost) independent from specific requirements or business decisions. This can be tasks such as setting up the deployment pipeline, learning a required programming language, setting up the test environment etc.

Keep the focus on the tasks that are not dependent on decisions from business or cus-

tomers, especially in the beginning of the project. The work done for these tasks is more likely to still be valuable for the project even after requirements changed.

Variants
The Task Separation is a very project specific practice. It depends on the Scrum Master and especially the PO to identify tasks that are not as requirement-specific as others. These tasks may vary from project to project.

Consequences
Benefits:

- Work results can be utilized even after requirements changed.
- Team stays motivated.
- Team can adapt more quickly to new requirements.

Liabilities:

- Additional effort for categorization of tasks.
- Depends on PO to judge which tasks are not tightly linked to specific requirements.

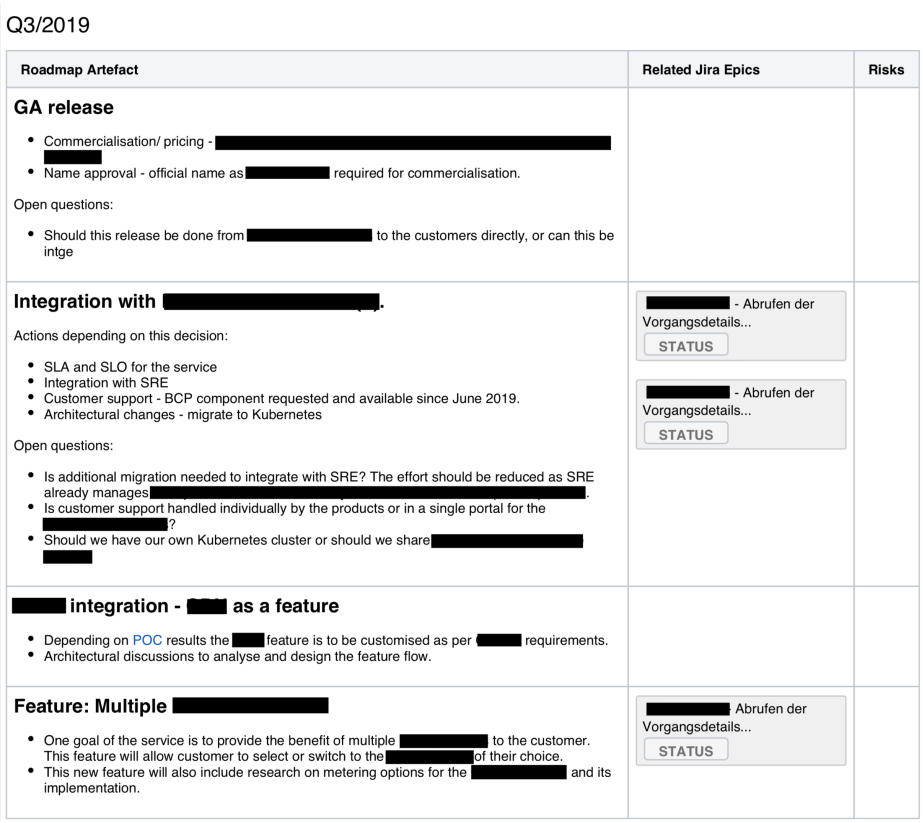
C.2.9. Further Methodology Pattern Candidates

ID	Name & Problem	Summary
M-9	PAIR PROGRAMMING **C-32**: *How to deal with lacking team cohesion at different locations?* **C-120**: *How to coordinate work across multiple time zones?* **C-134**: *How to deal with inefficient coordination meetings?*	Pair programming has been mentioned as a useful tool of regular use by interviewees from all three product areas. Especially for knowledge sharing and providing feature specific help across different locations and teams. The pair programming technique was used as a direct person to person coordination mechanism. This was enabled by video conference and screen sharing tools used at the organization. **Other Standards**: XP [4]

C.3. Documentation of Good Viewpoint Practices

C.3.1. Roadmap

Pattern Overview	
ID	V-1
Name	ROADMAP
Alias	
Summary	The Roadmap is a document maintained by the Product Owner. It intendeds to bridge the communication between business stakeholders and the development team. The Roadmap contains rows of 'Roadmap Artifacts'. The Product Owner also documents potential high-level risks.



Q3/2019

Roadmap Artefact	Related Jira Epics	Risks
GA release • Commercialisation/ pricing - [redacted] • Name approval - official name as [redacted] required for commercialisation. Open questions: • Should this release be done from [redacted] to the customers directly, or can this be intge		
Integration with [redacted]. Actions depending on this decision: • SLA and SLO for the service • Integration with SRE • Customer support - BCP component requested and available since June 2019. • Architectural changes - migrate to Kubernetes Open questions: • Is additional migration needed to integrate with SRE? The effort should be reduced as SRE already manages [redacted]. • Is customer support handled individually by the products or in a single portal for the [redacted]? • Should we have our own Kubernetes cluster or should we share [redacted]	[redacted] - Abrufen der Vorgangsdetails... STATUS [redacted] - Abrufen der Vorgangsdetails... STATUS	
[redacted] integration - [redacted] as a feature • Depending on POC results the [redacted] feature is to be customised as per [redacted] requirements. • Architectural discussions to analyse and design the feature flow.		
Feature: Multiple [redacted] • One goal of the service is to provide the benefit of multiple [redacted] to the customer. This feature will allow customer to select or switch to the [redacted] of their choice. • This new feature will also include research on metering options for the [redacted] and its implementation.	[redacted] Abrufen der Vorgangsdetails... STATUS	

Figure C.1.: Roadmap quarter as maintained by the interviewee PO2

Example

Interviewee PO2 of Team2 is maintaining a Roadmap for Product B. The Roadmap is structured quarterly and contains the upcoming four quarters. A screenshot of one quarter is shown in Figure C.1.

Context
Stakeholders have to be informed about what the other stakeholders are doing. The business side does not want to have to look inside the issue tracking tool which the development side is using. The development side wants to have an overview of the project's horizon.

Problem
C-44: *How to deal with communication gaps with stakeholders?*

Forces

- The Product Owner has to decide which information to pass from the business stakeholders to the development team and vice versa.
- The Product Owner has to ensure transparency and prioritization of overall goals to both, the business side and the development side.

Solution
The Roadmap is a document maintained by the Product Owner. It intendeds to bridge the communication between business stakeholders and the development team. The Roadmap contains rows of "Roadmap Artifacts". Each Roadmap Artifact can be linked to a concrete Epic. The Product Owner also documents potential high-level risks (i.e., not development risks but risks like architectural change or timeline risks). This document is intended to inform all the stakeholders about the current status of the project, upcoming features, and potential risks. Therefore, it facilitates the Product Owner in his task to coordinate and communicate the requirements of the business to the team, and to inform the business about the project status.

Variants
The Roadmap should be accessible to all stakeholders. Depending on the project the Roadmap can be structured quarterly or using other time spans.

Consequences
Benefits:

- The PO has a place to document the project status.
- Both sides (business and development) can find links to the respective other side's resources (e.g., Epic links, requirements documentation).
- Prioritization can be clearly indicated.

Liabilities:

- Adds another document that has to be maintained and updated.

Data Collection

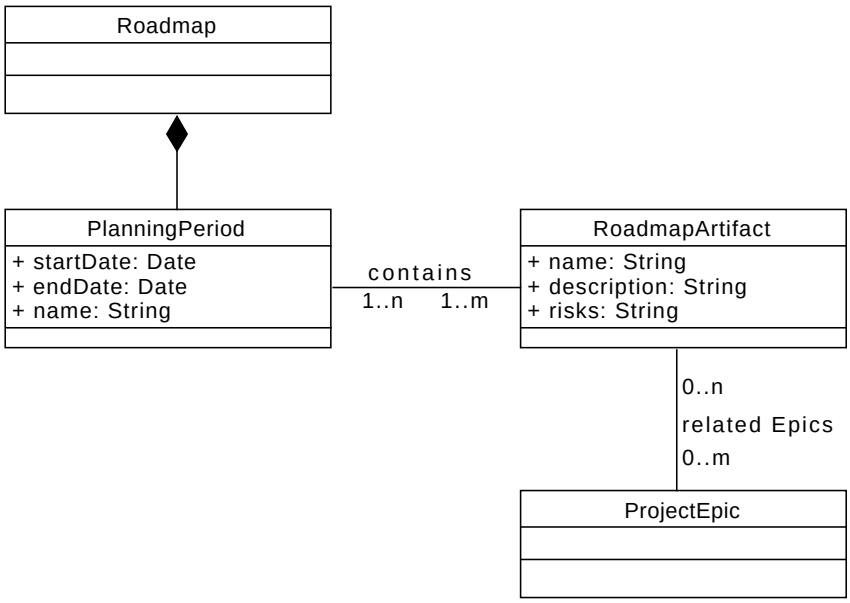


Figure C.2.: The data model for the Roadmap

The Roadmap is maintained by the Product Owner in such a way, that all stakeholders of the project / product can access it. It is updated whenever progress on the Roadmap has been achieved or a new period is added on the horizon. The data model for the Roadmap is shown in Figure C.2.

C.3.2. Task Dependency Mapping

Pattern Overview	
ID	V-2
Name	TASK DEPENDENCY MAPPING
Alias	
Summary	The Task Dependency Mapping is created to visualize the dependencies between sub-tasks of a task. It helps to efficiently utilize the whole team's capacity to resolve an urgent issue, or an issue that is blocking other tasks from progressing. It splits the issue in sub-tasks and provides an overview of dependencies, work allocation to the team members, and necessary handovers.

Example
Team5 used a Task Dependency Mapping after a team member went on holiday. The other members had to resolve his current task very quickly because they were depending on it. The mapping they created visualizes which sub-tasks of the job can be separated and assigns individual people to those sub-tasks. Further, the mapping visualizes the dependencies between the sub-tasks and who has to handover his work or results to whom, in order to quickly resolve the issue.

Context
An (unexpected) dependency between tasks is discovered during a sprint that is blocking team members from continuing their work.

Problem
C-95: *How to rapidly deliver a necessary patch?*
C-121: *How to deal with not being able to physically sit together in distributed teams?*
C-122: *How to deal with unexpected dependencies?*

Forces
- Not every dependency can be foreseen before a sprint is started.
- If something is very urgent it is not easy to use the whole capacity of the team efficiently for resolving that issue.

Solution

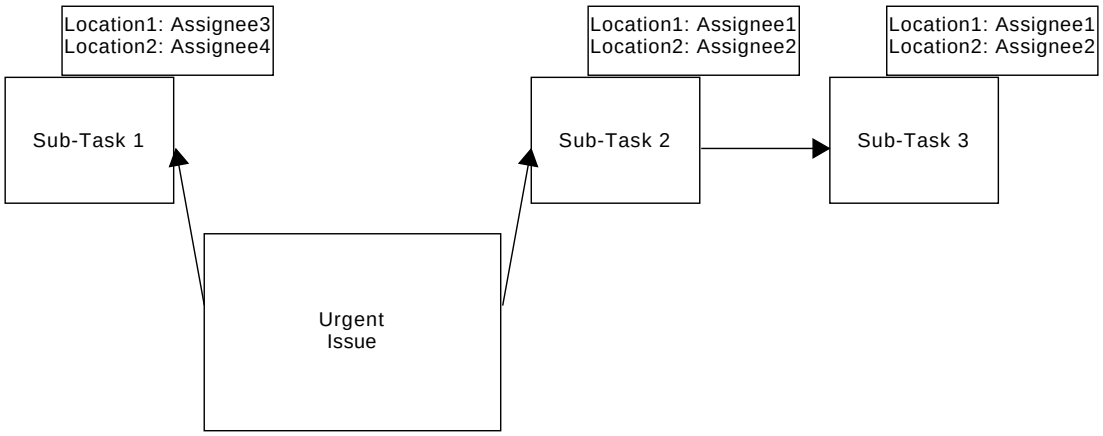


Figure C.3.: Task Dependency Mapping

To deal with a blocking dependency between tasks or an urgent issue, divide the issue into sub-tasks. Assign the sub-tasks to different team members to parallel the work as much as possible. Dependencies and needed handovers between sub-tasks have to be visualized. This helps team members to know whom to inform about the progress and enables development across time zones. A generic version of a dependency mapping is shown in Figure C.3. The status of sub-tasks is indicated by color-codes (coloring the sub-tasks in different colors).

Variants
The Mapping can be created on any board that is accessible to all involved team members, e.g., an online collaboration tool.
It can also be useful to apply the Task Dependency Mapping together with the **M-4**: FOLLOW THE SUN practice, to visualize handover scenarios of (depending) tasks across time zones.

Consequences
Benefits:

- The capacity of the team is used efficiently to get blocker dependencies out of the way.
- All depending / blocked team members can help to work on the blocker.
- Organizational things (e.g., order and handovers) are clearly defined.

Liabilities:

- Involves detailed upfront planning by Product Owner and Scrum Master.

See Also

- **M-4**: FOLLOW THE SUN

Data Collection

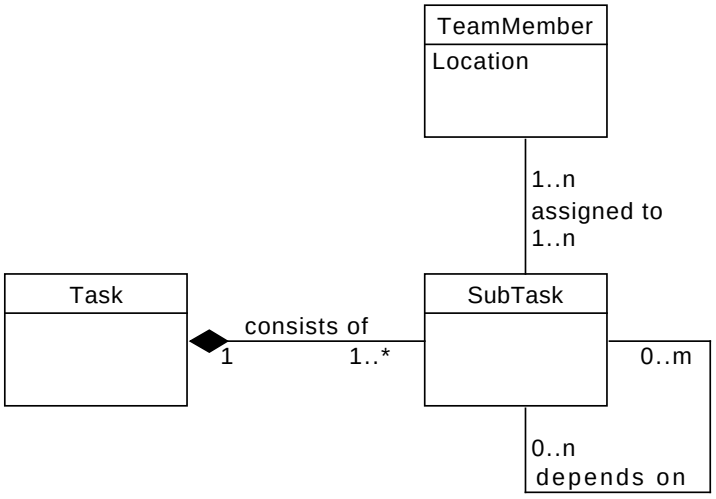


Figure C.4.: Data Model for the Task Dependency Mapping

The Product Owner and Scrum Master collect the necessary information about how the task can be split into sub-tasks. They assign the sub-tasks to developers in such a way that with the help of handovers the working time is maximized. The data model for the Task Dependency Mapping is shown in Figure C.4.

C.3.3. Milestone Planning Board

Pattern Overview	
ID	V-3
Name	MILESTONE PLANNING BOARD
Alias	
Summary	The Milestone Planning Board is used by the Product Owners during the Milestone Planning Meeting. It is used to explain and document the planned work and the estimates by the teams.

Example
The product area of AC1 is planning and distributing their work across teams using the **CO-12**: MILESTONE PLANNING MEETING. To keep track of who is planning to do what, how the teams estimate the effort, and how work could be reallocated, the Product Owners document the outcomes of the Milestone Planning Meeting in an Excel implementation of the Milestone Planning Board.

Context
The Milestone Planning Meeting is implemented in the organization to plan work in projects with multiple teams. The Product Owners discuss what they plan to implement in the upcoming sprints and explain how their teams estimate the effort.

Problem
C-1: *How to coordinate multiple agile teams that work on the same product?*
C-124: *How to implement scaled agile methodologies?*
C-125: *How to plan work and track progress?*

Forces

- Work planning across multiple teams is complicated and time consuming.
- The Product Owners have to know what is going on in other teams and document it.
- The Milestone Planning Meeting is only attended by Product Owners. Other project members should be able to review the outcomes of the Milestone Planning Meeting.

Solution

Owner	Feature	Issue link	Area/SIG/WG	Team1	Team2	Team3
PO1	Enable using a default messaging middleware other than NATSS	https://github.com/some-project/issues/7845	area/installation, area/documentation			3,2
...						

Figure C.5.: Milestone Planning Board

The board is organized in a table format as shown in Figure C.5. For each feature that a Product Owner wants to implement for the upcoming theme, a row is added to the Milestone Planning Board. Each row contains information about the responsible Product Owner, the feature the Product Owner wants to implement in the upcoming theme, a link to the development ticket, area categorization of the feature, and the estimates of the teams that will be involved in the development of this feature.

Variants
The board can be maintained in any feasible way that is accessible to team members. The observed organization created the board using an Excel file, but it can also be done with other tools.

Consequences
Benefits:

- All Product Owners can see what other teams are doing.
- The work estimation and features are documented.
- Other stakeholders can review the outcomes of the Milestone Planning Meeting.

Liabilities:

- The document can get pretty large very quickly, because all teams and all features for the upcoming theme are listed.

See Also

- **CO-12**: MILESTONE PLANNING MEETING

Data Collection
The data is filled out by the Product Owners. They collect the necessary data in advance of the Milestone Planning Meeting together with their teams. They prepare the Milestone Planning Board before the actual meeting takes place.

C.3.4. Sailboat Retrospective

Pattern Overview	
ID	V-4
Name	SAILBOAT RETROSPECTIVE
Alias	
Summary	The Sailboat Retrospective is a visualization practice to visualize the *Wind*, *Anchors*, and *Rocks* that the team encountered during the last sprint.

Example

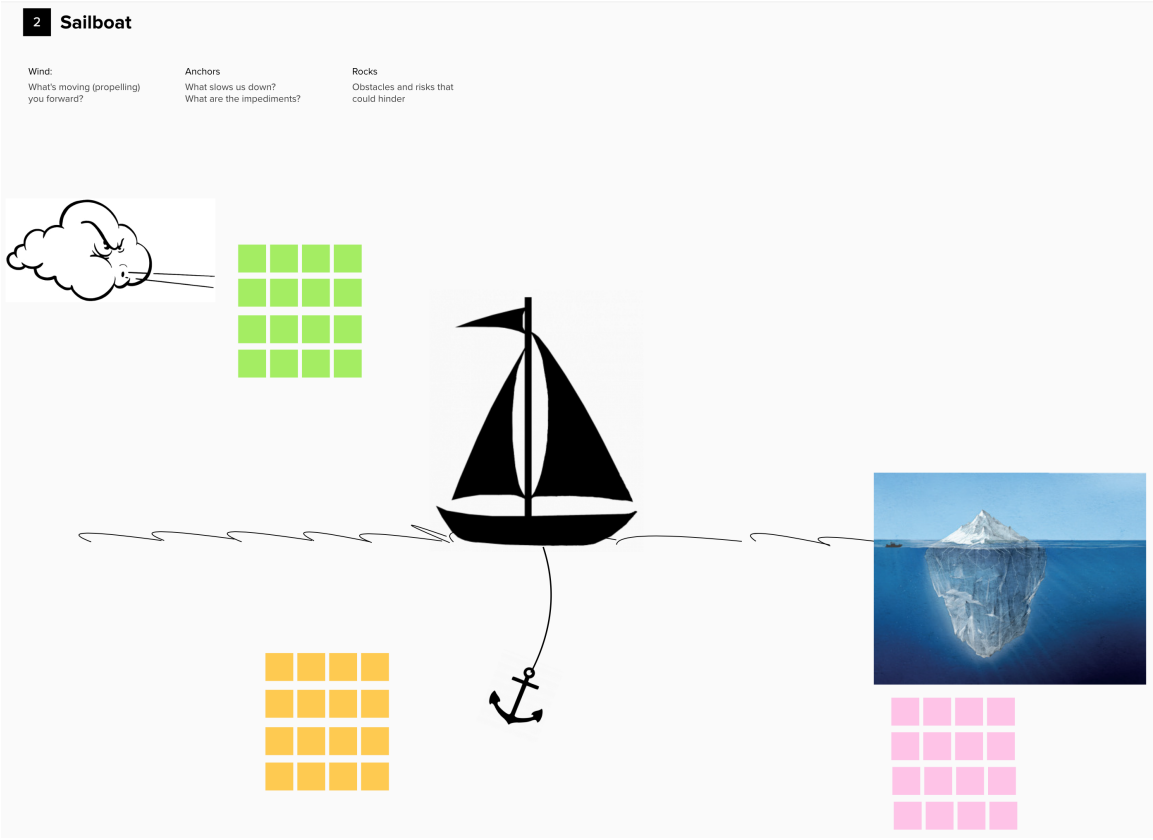


Figure C.6.: Sailboat Retrospective Template used by interviewee AC1

Figure C.6 shows an example template of the Sailboat Retrospective that is used by interviewee AC1.

Context
Scrum teams run a Sprint Retrospective after each Sprint. In the Sprint Retrospective the team members reflect on their work processes and discuss how they could improve their collaboration.

Problem
C-39: How to establish a culture of continuous improvement?
Could also be seen as a revised solution for **AP-1**: RANTROSPECTIVE.

Forces

- People tend to forget about the outcomes without documentation.
- Without distinction between types of points that are raised in a retrospective, it is hard to deduce suitable actions to improve.

Solution
The template for the Sailboat Retrospective that is used by interviewee AC1 can be seen in Figure C.6. The visualization should contain the following three areas to place (virtual) post-its on:

- **Wind**: What's moving (propelling) you forward?
- **Anchors**: What's slowing you down? What are the impediments?
- **Rocks**: Obstacles and risks that could hinder

Variants
The way how the Sailboat visualization is created can be varied. It can be created physically on a whiteboard with real post-its. But it can also be maintained in an online collaboration tool to use it with distributed teams.

Consequences
Benefits:

- The team reflects on processes and possible improvements.
- The template is not as tempting for **AP-1**: RANTROSPECTIVE as the normal scheme for retrospectives.

See Also

- **AP-1**: RANTROSPECTIVE

Data Collection
The Sailboat visualization is created collectively by the team during the Sprint Retrospec-

tive Meeting. Every team member can add (virtual) post-its to the three categories to express what they think about the previous sprint. The host of the Retrospective (Scrum Master or Agile Coach) takes care of archiving the document afterwards and creating the resulting items / tasks.

C.3.5. Team Homepage

The viewpoint practice **V-5**: TEAM HOMEPAGE was already presented in the findings in Section 4.4.7.

C.3.6. Persona

Pattern Overview	
ID	V-6
Name	PERSONA
Alias	
Summary	It is important to describe the target Personas as early as possible in the project. The Persona descriptions should be documented and be accessible for every stakeholder.

Example
The Personas for Product E are documented in the project's wiki pages and are also physically hanging at the office walls in form of large posters.

Context
The development team should know as early as possible whom they are actually targeting with their project. The better the target user is described, the better the product can be designed for this user.

Problem
C-15: *How to elicit and refine requirements of end users?*

Forces

- It is hard to get a common understanding of the actual user of the product.
- Products cannot be designed properly if the end user is not known to the development team.

Solution
Create Persona documentations and make them accessible to all stakeholders of the project. A Persona with the required fields is illustrated in Table C.1. The Persona should contain a caption that summarizes the described character in one sentence, a name to 'personalize' the Persona, and the role name. Further, the Persona should describe which expectations the described character has regarding the developed product, and which viewpoint or access-level the described character will have on the product.

"I make sure [Product] is up and running for the teams"

Name	Cassie
Role	Customer System Administrator
Responsibilities and Tasks	- Responsible to setup and configure [Product Name]: deploy container (maybe), connects to identity management - She sets up the tunneling between [Product Name], other applications, and [Product Name] - ...
Expectations	- It must be easy to connect [Product Name] to [Product Name] - I want to use the same user database / LDAP as for my [Product Name] and [Product Name] so that my developers don't have to create another user - There are templates / How-To's on how to connect [Product Name] with [Product Name] Applications - ...
Access & Rights	• log into the [Product Name] with [...] credentials • initiate the provisioning of [Product Name] from Portal • (or initiates deployment manually) • view [Product Name] instance on Portal • ...
Status	<Validation with customers, partners, other internal teams>

Table C.1.: An example Persona documentation from the case organization

Variants
The Persona documentation can be distributed in various ways: in the online wiki, physically on walls in the form of posters, or in other ways.

Consequences
Benefits:

- All stakeholders have a common vision of the end user Persona.
- The product can be designed targeted at the Persona.

Liabilities:

- The Persona might be subject to change.

Data Collection
The Personas are described, created, and maintained by the Product Owner in close collaboration with the customer of the product / project. This ensures accurate descriptions of the final users.

C.4. Documentation of Bad Practices

C.4.1. Rantrospective

Pattern Overview	
ID	AP-1
Name	RANTROSPECTIVE
Alias	
Summary	The standard sprint retrospective template is using 3 columns with the questions “What I don’t like”, “What I like”, and “What should stay”. These questions lead the team to rant and show their frustration, rather than coming up with constructive points.

Context
The team has to reflect on what did not go well in the past sprint in order to improve in the future.

Problem
C-39: *How to establish a culture of continuous improvement?*

Forces

- Retrospectives are necessary to find possibilities to improve the work.
- However, the setting of the Sprint Retrospective is tempting for people to rant on the previous sprint and to only focus on negative things.

General Form
The retrospective is built around a template of three columns with one question each. Every team member has to give an answer to all those three questions. The questions are:

- “What I don’t like”
- “What I like”
- “What should stay”

Consequences
Liabilities:

- The negatively phrased question tempts team members to really rant on certain things and show all of their frustration.
- The questions do not encourage team members to think of ways to improve on the problems.

Revised Solution
Phrase the questions more intelligently:

- "Things I would like to improve"
- "Things which went well"

This points the focus towards possible improvements rather than negative problems.

See Also

- The **V-4**: SAILBOAT RETROSPECTIVE is a way to run the retrospective more positively minded.

C.4.2. Demo Driven Development

The anti-pattern candidate **AP-2**: DEMO DRIVEN DEVELOPMENT was already presented in the findings in Section 4.4.9.

C.4.3. Don't Use Agile as Magic Bullet

Pattern Overview	
ID	AP-3
Name	DON'T USE AGILE AS MAGIC BULLET
Alias	
Summary	Agile methodologies are the go-to methodologies for new software development projects nowadays. However, in some cases they are not the best choice for the project and other methods should be considered as well.

Example
At the case organization teams are free to choose which agile methodology to use for their projects. However, SA2 mentioned that agile methodologies might not be the best way to manage some of the organizations projects.

Context
Agile methods are the go-to methodology in software companies for new projects. Teams usually get to choose their own methodology, but within the frame of agility.

Problem
C-6: *How to deal with incorrect practices of agile development?*
C-124: *How to implement scaled agile methodologies?*

Forces

- At the moment, besides agile often no other ways to run projects are considered by many managers.
- Not every project is best managed with agile methods.

General Form
When a new project is set up, the management usually gives a frame on how the project should be managed. At the case organization, the frame was that the methodology of choice should be agile.

Consequences
Benefits:

- All new projects are using development methodologies from a defined (agile) scope.

Liabilities:

- There might be better, non-agile ways to manage a specific new project.
- Agile methods are not the best way to manage new projects if team members are not

educated regarding agile values and practices.

Revised Solution
For some projects (e.g., when the scope and requirements are known and fixed up front) traditional methodologies might be worth a try. Further, if agile methods are applied without extensive coaching for values and practices, often projects might run smoother using the acquainted methodologies.

C.4.4. Too High-Level Scrum of Scrums

Pattern Overview	
ID	AP-4
Name	TOO HIGH-LEVEL SCRUM OF SCRUMS
Alias	
Summary	In large multi-team setup it is hard to do informative Scrum of Scrums meetings in a feasible time-span. Often the discussions are too high-level to be relevant for developers.

Example
The teams of Product E are doing Scrum of Scrums meetings. However, interviewee D2 mentioned that those meetings are not relevant to his work because the discussions were to high-level.

Context
Teams working together on a project have to communicate what they are working on and coordinate their work.

Problem
C-83: *How to deal with lacking knowledge of another team's activities?*

Forces

- The normal meetings of agile teams (e.g., the Scrum meetings) are too detailed for people from other teams. They do not need that much detail.
- To make higher-level meetings (like Scrum of Scrums) more interesting for all participants, people are tempted to make them more abstract and less in-depth.

General Form
The teams of a project setup a Scrum of Scrums meeting. In this meeting the teams or representatives of teams discuss what their individual are currently doing and which impediments they are facing. At the case organization, however, interviewees reported that the Scrum of Scrums meeting was too high-level for technical stakeholders like developers. Thus, the Scrum of Scrums meeting does not fulfill its intention.

Other Standards

- Paasivaara et al. [56] also report that 'general purpose' Scrum of Scrums meetings are not interesting and relevant to all participants anymore.

C.5. Documentation of Principle Candidates

C.5.1. Prerequisites to Form Autonomous Teams

Pattern Overview	
ID	P-1
Name	PREREQUISITES TO FORM AUTONOMOUS TEAMS
Alias	
Summary	To build an autonomous and focused team, the three prerequisites of this principle have to be fulfilled: 1. The team needs to trust the PO and should never bypass her/him in stakeholder communication or decisions. 2. The team has to have a common understanding of why agile methods are used within the team. 3. Measurable key performance indicators have to be defined up front, to be able to constantly check if the team is working towards the right direction.
Type	Coordination
Binding Nature	Mandatory

Example
The Team1 of the case organization was set up from scratch in 2018. Almost the entire team was new hires at that time. The software architect SA1 described that they had to implement this principle first to get started.

Context
A team or a project and its teams were just set up. The team members or most of the team members have not worked in this exact setup previously.

Problem
C-16: *How to establish self-organization?*
C-75: *How to form and manage autonomous teams?*

Forces

- A new project is created by top-down strategy decisions, so the prerequisites have to be implemented by the team members on their own after project setup.
- After the creation of a new team, people do not have s common understanding of how communication and coordination can work for this team.
- It is hard to determine progress of the team, especially in the beginning.

Consequences
Benefits:

- The team is autonomous in the sense that the PO can effectively control the information exchange between stakeholders and development team.
- The progress of the team can be clearly measured and improved.
- People are more engaged in improving the team's agile practices if they understand why they are applied.

Liabilities:

- It is often hard to define good, measurable key performance indicators. They are very project and team specific.

C.5.2. Spread Knowledge

Pattern Overview	
ID	P-2
Name	SPREAD KNOWLEDGE
Alias	
Summary	To avoid division of knowledge within teams, the work topics should be rotated among team members in such a way, that no single person develops a component completely alone. This ensures knowledge is spread across multiple people.
Type	Knowledge Management
Binding Nature	Recommended

Context
The team is working on multiple topics at the same time. All topics are important for the progress of the project.

Problem
C-100: *How to deal with division of knowledge within and between teams?*

Forces

- Teams should be able to work independently / autonomously (to a certain extent). However, the knowledge should not be only in one team or at one person.
- The team wants to progress as fast as possible but also avoid knowledge silos.

Variants
The rotation schedule can be varied. It should not be too long to avoid too much knowledge centralization. This can also be applied using teams and components. The teams can rotate the work on components of their project in a regular schedule.

Consequences
Benefits:

- Knowledge about components and solutions is spread across the team members.
- Reduces the danger of losing knowledge in case of personnel change or unexpected events.

Liabilities:

- Reduces the passion of the teams and individual developers for their work. There is no component "ownership" anymore.
- Reduces continuity in individual work. Requires familiarization with the topic on every sprint beginning.

Bibliography

[1] Christopher Alexander. *A Pattern Language: Towns, Buildings, Construction*. New York, USA: Oxford university press, 1977.

[2] Scott W. Ambler. *More Process Patterns: Delivering Large-scale Systems Using Object Technology*. New York, USA: Cambridge University Press, 1999.

[3] Scott W. Ambler. *Process Patterns: Building Large-scale Systems Using Object Technology*. New York, USA: Cambridge University Press, 1998.

[4] Kent Beck. "Embracing Change with Extreme Programming". In: *IEEE Computer* 32.10 (1999), pp. 70–77.

[5] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. *Manifesto for Agile Software Development*. 2001. URL: `https://agilemanifesto.org` (visited on 05/21/2019).

[6] Mike Beedle, James O. Coplien, Jeff Sutherland, Jens C. Østergaard, Ademar Aguiar, and Ken Schwaber. *Essential Scrum Patterns*. 2010. URL: `https://www.hillside.net/plop/2010/papers/beedle.pdf` (visited on 06/13/2019).

[7] Mike Beedle, Martine Devos, Yonat Sharon, Ken Schwaber, and Jeff Sutherland. "SCRUM: A Pattern Language for Hyperproductive Software Development". In: *Pattern languages of program design 4*. Ed. by James O. Coplien, John M. Vlissides, and Neil Harrison. Boston, USA: Addison-Wesley, 1999.

[8] Izak Benbasat, David K. Goldstein, and Melissa Mead. "The Case Research Strategy in Studies of Information Systems". In: *MIS Quarterly* 11.3 (1987), pp. 369–386.

[9] Saskia Bick, Alexander Scheerer, and Kai Spohrer. "Inter-Team Coordination in Large Agile Software Development Settings: Five Ways of Practicing Agile at Scale". In: *Proceedings of the Scientific Workshop Proceedings of XP2016*. Edinburgh, UK: ACM, 2016, 4:1–4:5.

[10] Saskia Bick, Alexander Scheerer, Kai Spohrer, Thomas Kude, and Armin Heinzl. "Software Development in Multiteam Systems : A Longitude Study on the Effects of Structural Incongruencies on Coordination Effectiveness". In: *eProceedings of the 9th International Research Workshop on Information Technology Project Management*. Auckland, NZ: AISeL, 2014, pp. 49–56.

[11] Kurt Bittner. *The Nexus framework for scaling scrum*. Ed. by Patricia Kong and David West. Boston, USA: Prentice-Hall, 2018.

[12] Barry W. Boehm. "A Spiral Model of Software Development and Enhancement". In: *SIGSOFT Softwware Engineering Notes* 11.4 (1986), pp. 14–24.

[13] Barry W. Boehm and Richard Turner. "Management challenges to implementing agile processes in traditional development organizations". In: *IEEE Software* 22.5 (2005), pp. 30–39.

[14] Teodora Bozheva and Maria E. Gallo. "Framework of Agile Patterns". In: *Software Process Improvement*. Ed. by Ita Richardson, Pekka Abrahamsson, and Richard Messnarz. Berlin, DE: Springer, 2005, pp. 4–15.

[15] Sabine Buckl, Florian Matthes, Alexander W. Schneider, and Christian M. Schweda. "Pattern-Based Design Research – An Iterative Research Method Balancing Rigor and Relevance". In: *8th International Conference on Design Science Research in Information Systems*. Berlin, DE: Springer, 2013, pp. 73–87.

[16] Erran Carmel, Yael Dubinsky, and Alberto Espinosa. "Follow The Sun Software Development: New Perspectives, Conceptual Foundation, and Exploratory Field Study". In: *2009 42nd Hawaii International Conference on System Sciences*. Waikoloa, USA: IEEE, 2009, pp. 1–9.

[17] Lawrence Chung and Julio C. S. do Prado Leite. "On Non-Functional Requirements in Software Engineering". In: *Conceptual Modeling: Foundations and Applications*. Berlin, DE: Springer, 2009, pp. 363–379.

[18] Alistair Cockburn and Jim Highsmith. "Agile Software Development: The People Factor". In: *IEEE Computer* 34.11 (2001), pp. 131–133.

[19] James O. Coplien. "A Generative Development-process Pattern Language". In: *Pattern Languages of Program Design*. Ed. by James O. Coplien and Douglas C. Schmidt. New York, USA: ACM Press/Addison-Wesley, 1995, pp. 183–237.

[20] James O. Coplien. *Software patterns: Management briefs*. Cambridge, UK: Cambridge University Press, 1996.

[21] James O. Coplien and Neil B. Harrison. *Organizational Patterns of Agile Software Development*. Vol. 1. Upper Saddle River, USA: Prentice-Hall, 2004.

[22] Daniela S. Cruzes and Tore Dybå. "Recommended Steps for Thematic Synthesis in Software Engineering". In: *2011 International Symposium on Empirical Software Engineering and Measurement*. Banff, CA: IEEE, 2011, pp. 275–284.

[23] Kim Dikert, Maria Paasivaara, and Casper Lassenius. "Challenges and success factors for large-scale agile transformations: A systematic literature review". In: *Journal of Systems and Software* 119 (2016), pp. 87–108.

[24] Torgeir Dingsøyr, Tor E. Fægri, and Juha Itkonen. "What Is Large in Large-Scale? A Taxonomy of Scale for Agile Software Development". In: *International Conference on Product-Focused Software Process Improvement*. Cham, DE: Springer, 2014, pp. 273–276.

[25] Torgeir Dingsøyr and Nils B. Moe. "Research Challenges in Large-scale Agile Software Development". In: *SIGSOFT Software Engineering Notes* 38.5 (2013), pp. 38–39.

[26] Torgeir Dingsøyr and Nils B. Moe. "Towards Principles of Large-Scale Agile Development". In: *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation*. Cham, DE: Springer, 2014, pp. 1–8.

[27] Torgeir Dingsøyr, Nils B. Moe, Tor E. Fægri, and Eva A. Seim. "Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation". In: *Empirical Software Engineering* 23.1 (2018), pp. 490–520.

[28] Torgeir Dingsøyr, Knut Rolland, Nils B. Moe, and Eva A. Seim. "Coordination in multi-team programmes: An investigation of the group mode in large-scale agile software development". In: *Procedia Computer Science* 121 (2017), pp. 123–128.

[29] Tore Dybå and Torgeir Dingsøyr. "What Do We Know about Agile Software Development?" In: *IEEE Software* 26.5 (2009), pp. 6–9.

[30] Christof Ebert and Maria Paasivaara. "Scaling Agile". In: *IEEE Software* 34.6 (2017), pp. 98–103.

[31] Amr Elssamadisy. *Agile Adoption Patterns: A Roadmap to Organizational Success*. Addison-Wesley Professional, 2008.

[32] Alberto Espinosa, Javier Lerch, Robert Kraut, Eduardo Salas, and Stephen Fiore. "Explicit vs. implicit coordination mechanisms and task dependencies: one size does not fit all". In: *Team cognition: Understanding the factors that drive process and performance* (2004), pp. 107–129.

[33] Sallyann Freudenberg and Helen Sharp. "The Top 10 Burning Research Questions from Practitioners". In: *IEEE Software* 27.5 (2010), pp. 8–9.

[34] Durham Goode. *Scaling Mercurial at Facebook*. 2014. URL: `https://code.fb.com/core-data/scaling-mercurial-at-facebook/` (visited on 07/26/2019).

[35] Nina-Mareike 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 of Munich, 2019.

[36] Neil B. Harrison. "Advanced Pattern Writing - Patterns for Experiences Pattern Authors". In: *Pattern Languages of Program Design 5*. Ed. by Dragos Manolescu, Markus Voelter, and James Noble. Boston, USA: Addison-Wesley, 2006.

[37] Neil B. Harrison. "Organizational Patterns for Teams". In: *Pattern Languages of Program Design 2*. Ed. by John M. Vlissides, James O. Coplien, and Norman L. Kerth. Boston, USA: Addison-Wesley, 1996, pp. 345–352.

[38] Neil B. Harrison. "The language of shepherding". In: *Pattern languages of program design 5*. Ed. by Dragos Manolescu, Markus Voelter, and James Noble. 1999, pp. 507–530.

[39] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. "Design Science in Information Systems Research". In: *MIS Quarterly* 28 (2004), pp. 75–105.

[40] VersionOne Inc. *13th annual state of agile report*. 2019. URL: https://www.stateofagile.com/#ufh-i-521251909-13th-annual-state-of-agile-report/ (visited on 05/22/2019).

[41] Ivar Jacobson, Grady Booch, and James Rumbaugh. "The unified process". In: *IEEE Software* 3 (1999), pp. 96–102.

[42] Binnur Karabacak. "Survey and Analysis of Process Frameworks for an Agile IT Organization". Bachelor's Thesis. Technical University of Munich, 2017.

[43] Maryam Kausar and Adil Al-Yasiri. *Distributed Agile Patterns*. URL: http://stp872.edu.csesalford.com/distributedagilepatterns.html (visited on 06/17/2019).

[44] Maryam Kausar and Adil Al-Yasiri. "Distributed agile patterns for offshore software development". In: *12th International Joint Conference on Computer Science and Software Engineering (JCSSE)*. Songkhla, TH: IEEE, 2015.

[45] Allan Kelly. *Business patterns for software developers*. Chichester, UK: John Wiley & Sons, 2012.

[46] Maarit Laanti. "Characteristics and Principles of Scaled Agile". In: *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation*. Cham, DE: Springer, 2014, pp. 9–20.

[47] Phillip A. Laplante. *Antipatterns. Identification, Refactoring and Management*. Boca Raton, USA: Auerbach, 2006.

[48] Craig Larman and Bas Vodde. *Large-Scale scrum: More with LeSS*. Addison-Wesley Professional, 2016.

[49] Christian Lescher. "Patterns for Global Development: How to Build One Global Team?" In: *Proceedings of the 15th European Conference on Pattern Languages of Programs*. Irsee, DE: ACM, 2010, 6:1–6:6.

[50] Thomas W. Malone and Kevin Crowston. "The Interdisciplinary Study of Coordination". In: *ACM Computing Surveys* 26.1 (1994), pp. 87–119.

[51] Gerard Meszaros and Jim Doble. "A Pattern Language for Pattern Writing". In: *Pattern languages of program design 3*. Ed. by Robert C. Martin, Dirk Riehle, and Frank Buschmann. Boston, USA: Addison-Wesley, 1997, pp. 529–574.

[52] Matthew B. Miles, A. Michael Huberman, and Johnny Saldaña. *Qualitative Data Analysis. A Methods Sourcebook*. 4. ed. Los Angeles, USA: SAGE Publications, 2019.

[53] Ian Mitchell. *Agile Development in Practice*. London, UK: TamaRe House, 2016.

[54] Miguel J. Monasor, Aurora Vizcaíno, Mario Piattini, John Noll, and Sarah Beecham. "Towards a Global Software Development Community Web: Identifying Patterns and Scenarios". In: *2013 IEEE 8th International Conference on Global Software Engineering Workshops*. Bari, IT: IEEE, 2013, pp. 41–46.

[55] Helga Nyrud and Viktoria Stray. "Inter-team Coordination Mechanisms in Large-scale Agile". In: *Proceedings of the XP2017 Scientific Workshops*. Cologne, DE: ACM, 2017, 16:1–16:6.

[56] Maria Paasivaara, Casper Lassenius, and Ville T. Heikkilä. "Inter-team Coordination in Large-scale Globally Distributed Scrum: Do Scrum-of-scrums Really Work?" In: *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. Lund, SE: ACM, 2012, pp. 235–238.

[57] Rachel Potvin and Josh Levenberg. "Why Google Stores Billions of Lines of Code in a Single Repository". In: *Communications of the ACM* 59.7 (2016), pp. 78–87.

[58] Winston W. Royce. "Managing the Development of Large Software Systems: Concepts and Techniques". In: *Proceedings of the 9th International Conference on Software Engineering*. Monterey, California, USA: IEEE, 1987, pp. 328–338.

[59] Kenneth S. Rubin. *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Upper Saddle River, USA: Addison-Wesley, 2013.

[60] Per Runeson and Martin Höst. "Guidelines for conducting and reporting case study research in software engineering". In: *Empirical Software Engineering* 14.2 (2009), pp. 131–164.

[61] Nayan B. Ruparelia. "Software development lifecycle models". In: *ACM SIGSOFT Software Engineering Notes* 35.3 (2010), pp. 8–13.

[62] Alexander Scheerer. *Coordination in Large-Scale Agile Software Development. Integrating Conditions and Configurations in Multiteam Systems*. Springer, 2017.

[63] Alexander Scheerer, Tobias Hildenbrand, and Thomas Kude. "Coordination in Large-Scale Agile Software Development: A Multiteam Systems Perspective". In: *2014 47th Hawaii International Conference on System Sciences*. Waikoloa, USA: IEEE, 2014, pp. 4780–4788.

[64] Douglas C. Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects*. Chichester, UK: John Wiley & Sons, 2000.

[65] Ken Schwaber. "Scrum development process". In: *Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications*. 1995, pp. 117–134.

[66] Ken Schwaber. *The Nexus Guide*. 2018. URL: `https://www.scrum.org/resources/nexus-guide` (visited on 06/08/2019).

[67] Ken Schwaber and Jeff Sutherland. *The Scrum Guide*. 2017. URL: https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf (visited on 06/04/2019).

[68] Scrum.org. *The Scrum Framework Poster*. 2019. URL: https://www.scrum.org/resources/scrum-framework-poster (visited on 07/25/2019).

[69] ScrumPLoP. *Published Patterns*. 2019. URL: https://sites.google.com/a/scrumplop.org/published-patterns/home (visited on 08/08/2019).

[70] Jeff Sutherland, Neil Harrison, and Joel Riddle. "Teams That Finish Early Accelerate Faster: A Pattern Language for High Performing Scrum Teams". In: *2014 47th Hawaii International Conference on System Sciences*. Waikoloa, USA, 2014, pp. 4722–4728.

[71] Paul Taylor. "Capable, Productive, and Satisfied: Some Organizational Patterns for Protecting Productive People". In: *Pattern languages of program design 4*. Ed. by James O. Coplien, John M. Vlissides, and Neil Harrison. Boston, USA: Addison-Wesley, 1999.

[72] Isabelle Therrien and Erik LeBel. "From Anarchy to Sustainable Development: Scrum in Less than Ideal Conditions". In: *2009 Agile Conference*. Chicago, USA: IEEE, 2009, pp. 289–294.

[73] James D. Thompson. *Organizations in action*. New York, USA: McGraw-Hill, 1967.

[74] Ömer Uludağ, Nina-Mareike Harders, and Florian Matthes. *Documenting Recurring Concerns and Patterns in Large-Scale Agile Development*. 2019.

[75] Ömer Uludağ, Martin Kleehaus, Christoph Caprano, and Florian 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*. Stockholm, SE: IEEE, 2018, pp. 191–197.

[76] Antti Välimäki. *Pattern Language for Project Management in Global Software Development*. PhD Thesis. Tampere University of Technology, 2011.

[77] Andrew H. Van de Ven, Andre L. Delbecq, and Richard Koenig. "Determinants of Coordination Modes within Organizations". In: *American Sociological Review* 41.2 (1976), pp. 322–338.

[78] Tim Wellhausen and Andreas 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*. Irsee, DE: ACM, 2012, 5:1–5:9.

[79] Laurie Williams and Alistair Cockburn. "Agile software development: It's about feedback and change". In: *IEEE Computer* 36.6 (2003), pp. 39–43.

[80] Robert K. Yin. *Case study research: Design and Methods*. 5th ed. Los Angeles, USA: SAGE Publications, 2014.