FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatik

Empirical studies to identify best practices for addressing recurring concerns of product managers and product owners in large-scale agile development

Louis Leonardo Zschaler

FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatik

Empirical studies to identify best practices for addressing recurring concerns of product managers and product owners in large-scale agile development

Empirische Studien zur Identifikation bewährter Praktiken für die Adressierung wiederkehrender Herausforderungen von Produkt Managern und Ownern in Large - Scale Agile Development

Author:	Louis Leonardo Zschaler
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Ömer Uludağ, M.Sc.
Date:	December 16, 2019

d

I assure the single handed composition of this master thesis only supported by declared resources,

Munich, December 16, 2019 Louis Leonardo Zschaler

Abstract

Agile methods have been put to test for almost 20 years now resulting in mixed feelings whether or not they will revolutionize software development and other disciplines. Especially small teams were able to show highly positive results of agile methods, leading to an increased efficiency and a better work environment. Big companies become more and more interested in this trend, but while small teams already face several concerns in applying agile methods, large-scale agile methods come with a much higher complexity and a new set of concerns. This is why current literature points out more concerns about large-scale agile development than there are concepts for tackling these concerns.

In order to provide best practice solutions in form of patterns, the Chair of Software Engineering for Business Information Systems at the Technical University of Munich has developed a Large-Scale Agile Development Pattern Language. Patterns describe practice-proven solutions for recurring concerns. A pattern language defines the structure of a pattern in a given context, making it easily available to the industry. In the context of this thesis, 11 structured interviews with industry partners have been conducted to observe recurring concerns and best practices of Product Owner and Product Manager in Large-Scale Agile Development.

As a result 6 patterns and 30 pattern candidates have been documented in the Large-Scale Agile Development Pattern Language. Furthermore 23 new concerns have been identified with the industry partners. In total 58 concerns have been verified and their frequency shows how urgent each concern is. The observed patterns have been proven by at least three different industry partners. The value of the pattern candidates has to be evaluated further to see if they have the potential to be valuable to many more organizations with the same or similar concern.

Zusammenfassung

Agile Methoden wurden in den vergangenen 20 Jahren eingehend erprobt, wobei es immer noch gespaltene Gefühle drüber gibt, ob sie wirklich die Software Entwicklung und andere Disziplinen revolutionieren oder nicht. Vor allem kleine Teams haben einen erheblichen Mehrwert aus den agilen Methoden gezogen, was nicht nur zur Steigerung ihrer Effektivität geführt hat, sondern auch zu einem besseren Arbeitsumfeld.
Auch große Firmen interessierten sich nach und nach immer mehr für diesen Trend, aber dadurch, dass kleine Teams teilweise schon mit Herausforderungen zu kämpfen hatten, wurde es durch die Large-Scale Agile Methoden noch komplexer und führte zu noch mehr Herausforderungen. Aus diesem Grund lassen sich in der derzeitigen Literatur mehr Probleme, als Konzepte, die dabei helfen diese Probleme zu lösen, finden.
Um dem entgegenzuwirken hat der Lehrstuhl für Software Engineering for Business Information Systems an der Technischen Universität München (TUM) eine Large-Scale Agile Development Pattern Language entwickelt.
Patterns beschreiben eine Lösung für derzeitige Herausforderungen, die in der Praxis bereits getestet wurden. Eine Pattern Language definiert die Struktur eines entsprechenden Patterns in einem Context, der es einfacher für die Industrie macht, diesen anzuwenden.
Im Rahmen dieser Masterarbeit wurden elf Interviews mit verschiedenen Partnern aus der Industrie geführt, um derzeitige Herausforderungen zu beobachten und diese mit Hilfe der, von Product Ownern und Product Managern, verwendeten Lösungsansätze in Large-Scale Agile Development zu dokumetieren. Ergebnis dieser Feldforschung waren sechs patterns und 30 pattern candidates, die im Bereich Large-Scale Agile Development beobachtet wurden. Zusätzlich wurden, neben der Herausforderungen, die schon in der Literatur zu finden waren, 23 neue Herausforderungen dokumentiert. Alles in allem wurden 58 Herausforderungen verifiziert und die Häufigkeit zeigt, wie relevant diese sind. Die patterns müssen von mindestens drei verschiedenen Partnern bestätigt worden sein.
Der Wert eines pattern candidates muss noch weiter bestimmt werden, um zu evaluieren, ob sie einen Mehrwert für Firmen bringen können, die mit den gleichen Herausforderungen zu kämpfen haben.

Contents

Chapter 1

Introduction

The first chapter will introduce this thesis by showing the current status of Agile and Large-Scale Agile Development in the literature and the need for documenting best practices of concerns for Product Managers and Product Owners in a pattern language. Furthermore it will present the research objectives that are to be answered and how to approach them.

1.1 Motivation

For 20 years now agile methods have changed the way of collaboration for many teams and organizations. Iterative cycles of lean processes and delivering increments regularly result in more flexibility, more value to the customer and shorter time-to-market. According to a study conducted by The Standish Group International, Inc. in 2015, agile projects are almost four times more successful than waterfall projects. [1] Combined with the shorter time-to-market, this also leads to cost reduction. [1] These benefits has inspired more and more people to adapt agile principles and values. So far mostly small companies were able to transform their organization towards agile, but fast growing and changing markets with increasing competition is forcing large companies to get faster as well.

While agile frameworks originally were designed for small and self-organized teams with usually not more than 10 members [2], large companies with three-digit team sizes are faced with the challenge to scale agile methods to their needs. Large-Scale agile frameworks like LeSS [3] and SAFe® [4] try to enable large projects and companies to achieve the same benefits by scaling the traditional agile methods and having many agile teams work on the same product. The positive impact could be huge. The Chaos Report 2015 evaluated over 10,000 projects, showing that large size agile projects have six times the success rate as large size waterfall projects. [1]

However, there are many new challenges to be dealt with. The literature on dealing with challenges when becoming agile is full of publications [5] but the literature for large-scale agile development is scare. There are only a few positive examples like Lego [6] or Spotify [7] that managed a successful transformation. But for many documented challenges or concerns, there are no solutions or best practices to be found in the literature. [8]

A fast and efficient way to document solutions for recurring concerns are patterns. Uludağ et. al. defined a large-scale agile development pattern language to structure patterns in this very field. [9] It categorizes identified concerns, defines five different kinds of patterns and unlike other pattern language, it relates the documented patterns to different stakeholders. Especially Product Managers and Product Owners will have to face to concerns when managing not one but several agile teams. This new level of complexity will raise questions like how to distribute tasks, what to do with cross-team dependencies and how to keep a strong communication between all these self-organized teams.

1.2 Research Objectives

This thesis goal is to contribute patterns for Product Managers and Product Owners as part of the Large-Scale Agile Development Pattern Language (LSADPL). [9] In order to achieve that, the following research questions (RQ) will be answered within this paper.

Research Question 1: What are recurring concerns of Product Managers and Product Owners in Large-Scale Agile Development?

There are a great number of concerns when it comes to Large-Scale Agile Development. The problem is, that the list of publications evaluating these problems is rather short. That's why it is important to examine what the concerns for Product Owners and Product Managers are and how to evade them. Without these concerns, it would not be possible to identify different patterns, because they need to be applied to recurring concerns.

Research Question 2: What are best practices for addressing recurring concerns of Product Managers and Product Owners in Large-Scale Agile Development?

After identifying the concerns of Product Owners and Product Managers in RQ1 it is necessary to observe and evaluate different possible solutions. If you monitor a solution approach three or more times you can call it a pattern. Every solution, that is being used less than three times is no pattern yet, but is handled as a pattern candidate and can still give valuable information.

Research Question 3: Which anti-patterns should Product Managers and Product Owners avoid in large-scale agile development?

Analogical to RQ2 the third Research Question tries to identify several Anti-Patterns and interpret them. Just like Patterns they need to be applied at least three times to gain the label Anti-Pattern. If used less than three times they can not be called Anti-Pattern but Anti-Pattern Candidate.

1.3 Research Approach

The method used for this thesis is called Pattern-Based Design Research. The goal of this approach is "to balance rigor and relevance in Information System research". This approach focuses on patterns that are observed in practice. Pattern-Based Design Research consists of four activities: observe & conceptualize, pattern-based theory building & nexus instantiation, solution design & application and evaluation & learning (shown in figure 1.1 on page 4). This thesis scope will focus on the first two activities. [10]

The first activity represents the problem diagnosis. Good practices from different industries are observed and later documented with a specific pattern structure in order to get a set of pattern candidates. In this thesis eleven structured interviews are the primary source for finding pattern candidates. The structure of a problem diagnosis normally contains at least three or four elements: problem, solution, context and forces. Depending on the application domain the conceptualization might be more detailed. [10]

The second activity of pattern-based theory building & nexus instantiation describes how to evolve pattern candidates into a pattern language by bringing several patterns into relationship. If you observe the same use case at least three times it becomes a pattern. This pattern must then be set into relationship with already existing patterns to gain a structured set of patterns called pattern language. [10]

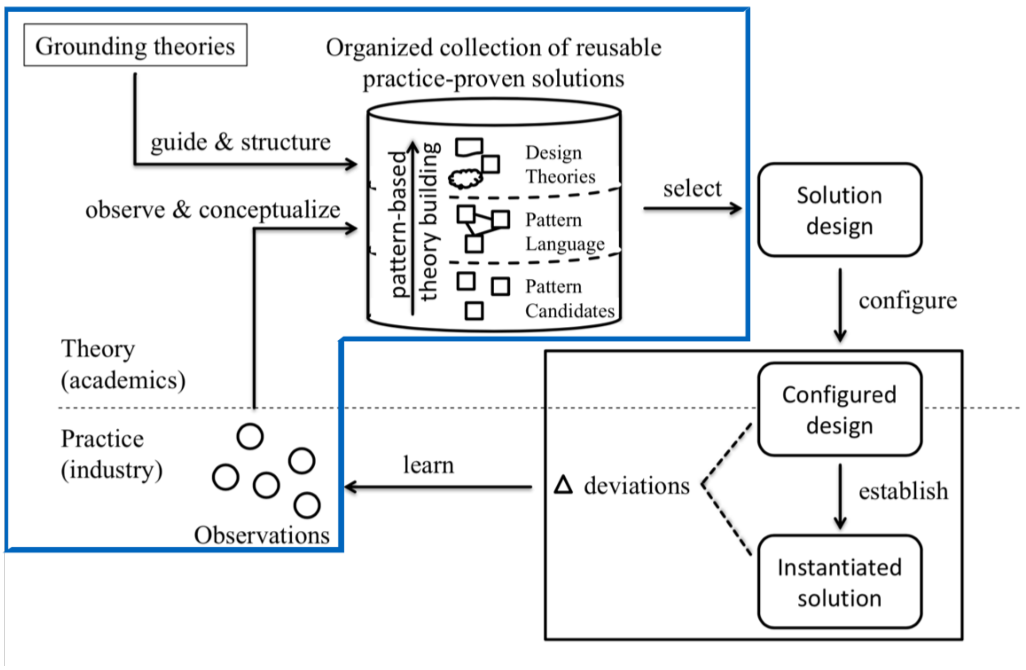


Figure 1.1: Pattern-based design research [10]

Evolving the pattern language then into a design theory would be the next step, but is out of scope for the purpose of this thesis. The results of the pattern language shall later be used in practice in large-scale agile environments in order to tackle an existing concern by implementing one or several related solutions.

The following thesis is structured as followed. Chapter two, the foundation discusses the fundamental elements, that were used for this work like Agile Software Development. To get a better insight in the different elements this sub-chapter is divided in four parts: agile value, agile principle, agile practices and agile frameworks, while agile frameworks mainly describes Scrum as an exemplary and well known agile frameworks. The second sub-chapter of the foundation will take a closer look on large-scale agile development, how it differs from the original agile development and give an insight on two large-scale frameworks. The third part will analyze the concept of patterns as a way of documenting best practice solutions.

Chapter three summarizes the most important and relevant work regarding agile software development, Large-Scale Agile Development and patterns.

Chapter four will describe the process of identifying recurring concerns and best practices and the methodology that was used. Chapter five discusses the results, that where gained by analyzing the interviews held and the last chapter summarizes the results of this thesis and an outlook on future work.

Chapter 2

Foundation

The following chapter will explain all relevant terms and concepts that build the foundation of this thesis. It will first explain the main elements of agile software development, how it was created, why there was a need for new development methods and illustrate one exemplary framework for agile software development. Based on that, this chapter aims to clarify what Large-Scale Agile Development means in this context and bring two frameworks as an example. In addition to that, this chapter will conclude in describing the use of patterns as a method for providing findings in a fast way.

2.1 Agile development

Traditional software development already brought out several development models based on the Software Development Lifecycle [11]. The basic concept of traditional software development is that development is a theoretical process, thus fully definable [12]. But for many use cases this approach does not apply, which created the need for a new method that builds upon an empirical process.

First approaches for agile software development started in the 1990s [12]. Kent Becks publication "eXtreme Programming" in 1999 [13] led to an increasing popularity in this matter. While several researchers worldwide published papers independently from each other, their findings had a lot in common. 17 of them composed the Agile Manifesto in 2001 which lays the foundation for today's agile methods and processes. These 17 authors came to the conclusion that this new approach requires a certain mindset. That is why processes and methods alone won't lead to the expected outcome. Agile values and principles are required as well.

2.1.1 Agile value

The group of 17 authors, also known as "The Agile Alliance", believed that agile methodologies require an environment that sets a higher focus on the people. Such an environment shall have people not working on a project because they have to, but because they want to see the project succeed. In order to create such a motivation, not based on monetary motives only, the Agile Manifesto describes the following values for agile software development: [14]

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

2.1.2 Agile principle

The second part of the Agile Manifesto describes agile principles. These principles are to be seen as guidelines and build together with the agile values, the foundation for all agile practices. Again the focus is very much on the customer and the people working in an agile environment. The Agile Manifesto lists 12 principles: [14]

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.

- Agile processes promote sustainable development. The sponsors, developers and users, should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity–the art of maximizing the amount of work not done–is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

2.1.3 Agile practices

Unlike principles, agile practices are concrete instructions that can be used for solving one or several problems. Practices define *how* to do something in a certain stage of the Software Development Lifecycle (SDLC). One example for agile practices can be the use of Story Cards [15]. This method describes a set of activities executed in an early stage of the SDLC as part of the requirement engineering. It aims to provide customers a way of documenting the requested behaviour of the target system to the developing team. Each functionality will be one story card. One of the benefits of this principle is the customer-centric approach from the very beginning of the project which is directly connected to the third agile value of the Manifesto [14]. So agile practices can guideline people to adopt agile values or principles and help understand their importance.
Other examples for agile practices can be test-driven development [16] or pair-programming [17].

2.1.4 Agile frameworks

Frameworks are taking in values, principles and practices to provide guidelines in approaching a certain goal. They provide a loose structure without being to detailed and leave room for other practices and tools to be included. Unlike methodologies, who are aiming to provide a holistic set of methods and guidelines to describe an entire life cycle, as in software development for instance the SDLC. [18]

Agile frameworks in particular usually have in common that they aim to keep the planning phase very short and reach a minimum valuable product in the implementation phase as fast as possible. A short planning phase is thereby reached with multiple iterations through the SDLC. This iteration makes the process *agile* because it passes the planning phase various times, giving developers and software engineers the possibility to constantly adapt the software to outer circumstances. This possibility for constant adaption is the most significant advantage over traditional (plan-driven) software development frameworks. Examples for traditional frameworks, among others, are the V-Model [19] or the waterfall model [20].

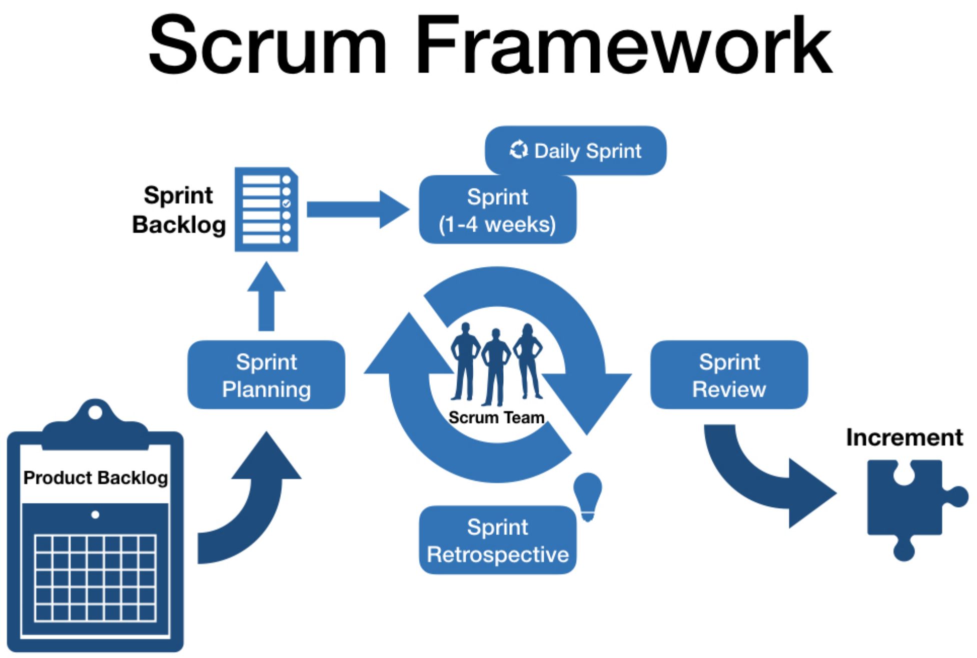


Figure 2.1: The Scrum Process [21]

Probably the most known and used agile framework is Scrum (shown in figure 2.1 on page 9). The official Scrum Guide, published by Ken Schaber and Jeff Sutherland, defines Scrum as "a framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value". [21] It is a lightweight framework which is easy to understand, but still difficult to master. The special aspect about it is, that its not just a process or simple technique. Instead it is a scheme, within which you can apply different processes and techniques to help improve the team, working environment and most important the product you are working on. [21]

To understand Scrum, its essential to know all the components included. Scrum consists of a team and their rules, roles, artifacts and events. Each team member has a particular role which is crucial not only for the success of a particular project but also for the use of this framework. [21] In total there are three different roles: Product Owner, Scrum Master and the Development Team.

The Product Owner is responsible for the whole project, meaning he enlarges the outcome of the work that the developing team accomplishes and also manages the Product Backlog to ensure, that it is transparent for everyone, meaning they understand the items within the Backlog and know what task they have to work on next. The Product Owners success depends on whether the team respects all decisions made and also on how every task is prescribed to keep the amount of dependencies low. [21]

The second role, the Scrum Master is responsible for the success of the team including helping everyone to understand the principles of Scrum and make them work. Also the Scrum Master acts as a broker between the inside team members and everyone outside, to make sure every interaction between them is high in value. [21]

The Development Team is organizing its own work, which means that they are self responsible. They work cross functional and each team member is equal. Scrum does whether recognize titles nor sub-teams or accountability to only one team member. Some may have specialized skills but these belong to the whole team and not just to an individual.

Beside those three roles the scrum framework also contains five main events: Sprint, Sprint Planning, Daily Scrum, Sprint Review and Sprint Retrospective. The most important event within Scrum is a Sprint. You can define it as a time-box, normally two to four weeks, in which a release candidate is produced and marked as done. [21] Before you start a Sprint the team comes together for a meeting also called the Sprint Planning, where you define the Sprint Backlog based on the Product Backlog. [22] To keep those goals in mind the team meets up every morning to the Daily Scrum, a short meeting where every team member gives a short recap of what they have achieved since the last, what they will do until the next meeting and if they have any difficulties that might require help or actions of an other team member. [23]

At the end of every sprint the team meets up for a Sprint Review and a Sprint Retrospective. The reason for doing the Review, is to go through the Increment and modify the Product Backlog if necessary. [21] The Sprint Retrospective helps the Scrum team to look back at their way of working and to reflect it, so that work can be improved in the next Sprint. This Retrospective is about three hours long and held before you start the next Sprint Planning. You don't only talk about the work and how to improve it, you also talk about the relationships within the team, the processes and tools. [21]

However, the scalability of agile methods can easily reach it's limits because they all talk about one agile team, where the maximum number of team members is usually considered as ten. The Scrum Guide suggested 7 ± 2 members per team based on the psychology paper "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information". [24] Later they increased the range making it 6 ± 3 members per team but the limit to 9 members stands. Other papers support this approach, stating that big teams are way less efficient. [2] Does this mean that large organizations with teams consisting of three-digit member amounts are not able to profit from agile methods?

2.1.5 Agile organizations

Agile development does not have to stop at team level. Agile principles and values can be applied within the whole organization. Thereby traditional hierarchical structures are usually to overcome (shown in figure 2.2 on page 12). In this context an organization does not have to equal a company. Agile organizations can also be applied to just a department or main section within a company. Growing companies like start-ups have to start building structures in their organization and can then easily decide where to go. Bigger companies with already existing structures have to go through a change process to become an agile organization, often referred to as Agile Transformation. The bigger the organization, the more complex, difficult and time-consuming the agile transformation.

2.2 Large-scale agile development

The first agile frameworks never considered scalability but with an increasing interest in agile development, new frameworks came up trying to bring agile methods to a new scale. The common answer over several large-scale agile frameworks is to split the project into several agile teams. This leads to a

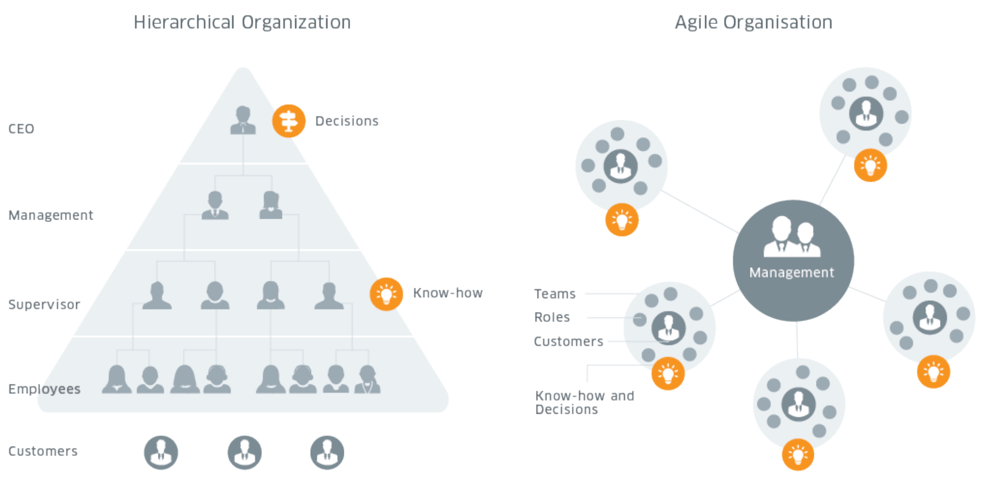


Figure 2.2: Agile Organization & Culture [25]

manageable controlling layer, focusing on the collaboration among the teams while there are agile practices within each team and around them. However, due to the enormous complexity that comes along with such big projects, the scope of Large-Scale agile frameworks goes far beyond just the team level. A new level of complexity brings new concerns for the organization. So applying agile frameworks to large companies has it's pros and cons.
Among the most common frameworks for Large-Scale Agile Development are LeSS and SAFe®.

2.2.1 LeSS

Agile methods were originally designed for small teams. Due to their success they inspired bigger companies to adapt those methods and use them for Large-Scale efforts. The fact, that they were designed for small teams means, that Large-Scale Agile Development comes with some concerns, the teams need to deal with. Craig Larman and Bas Vodde, founders of LeSS, separated it into two different frameworks: LeSS and LeSS huge. While the first one fits for two and up to eight teams, LeSS huge is for more than eight. [3] LeSS is scrum, applied to many different teams, so they are able to work cross-functional. Its about applying the elements and principles to a Large-Scale project and to keep it as easy as possible. Those teams, that work together have one common goal, meaning that everyone is responsible for the whole project, not only one part of it. LeSS is customer-centric, it focuses on the real problems and tries to find a solution.

LeSS does still have a Product Backlog, one Product Owner, one Product and one Sprint, regardless how many different teams are working on the project. [3] But besides that, each team has its own Scrum Master and Feature Team. Scrum has five different events and so does LeSS. All teams work within the same Sprint, meaning that they have a common Sprint Planning, Review and Retrospective, and not one each. But when it comes to Sprint Planning there is one huge difference from Scrum: you have two different phases, Sprint Planning One and Sprint Planning Two.

In Phase One the different teams, or a couple of representatives meet with the Product Owner to create the Product Backlog and clarify all items (shown in figure 2.3 on page 14). After that either each team has its own Sprint Planning, Phase Two, based on their Backlog where they are designing and planning the work they have to do, or if some teams have a Backlog that is quite similar, they do a multi-team Sprint Planning to identify their common projects to prevent doing tasks twice. At the end of every Sprint you have a Review and two Retrospectives. The difference to traditional scrum is, that you have a team Retrospective and an Overall Retrospective attended by Product Owner, Managers, Scrum Master and a couple of team representatives. [3]

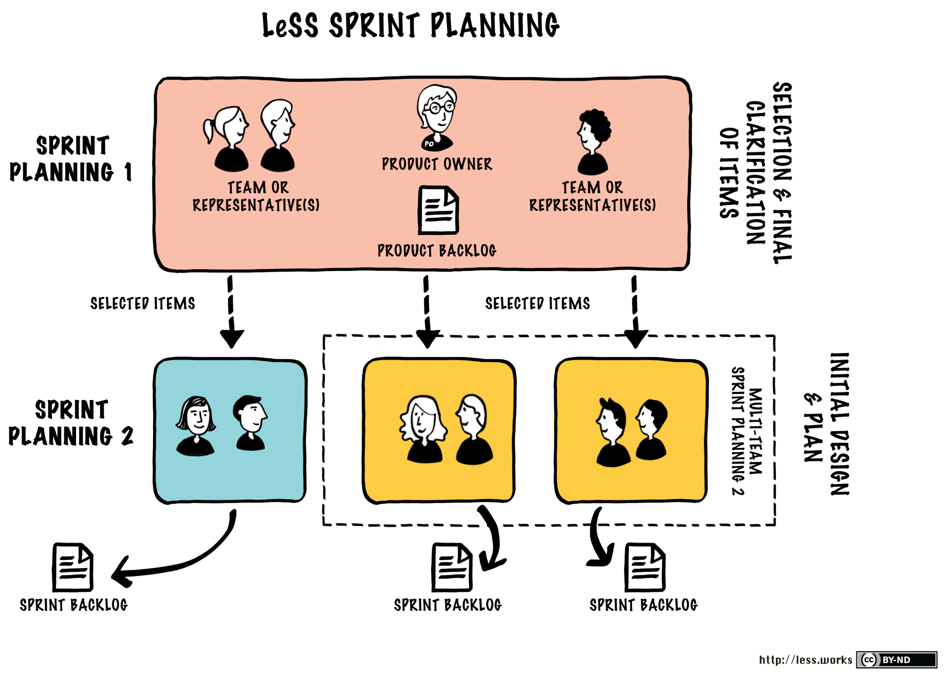


Figure 2.3: Less Sprint Planning [3]

2.2.2 SAFe

Like LeSS the Scaled Agile Framework (SAFe) is a framework, that can be used in Large-Scale Agile Software Development and combines methods like Kanban, Scrum and Extreme Programming with Lean Thinking and Lean Product Development, which makes it easier for huge enterprises to use agile principles. [26] SAFe provides not only artifacts but also collaboration and follows a centralized strategy. [27] There are three different levels when it comes to SAFe, while SAFe 4.0 has the highest scaling factor.

There a four layers that can be applied to SAFe. The first two, the team layer and the program layer are mandatory, the third and fourth, the Large Solution and the Portfolio layer, can additionally be applied, whether its needed or not. [4]

As mentioned before, SAFe has the highest scaling factor and can be used for the entire organization. [4] The bottom consists of the team layer. The Scrum that is used on this level is a bit different from the 'normal' Scrum, that was explained a bit earlier in this thesis, its based on XP and has between five to eight team members and a Scrum Master and Product Owner. [26] Each Product Owner has a very strong team focus and is responsible for about one or two teams besides supporting the Product Management.

Above the team you have the Program Level. This level contains the Agile Release Train (ART) and has between five and ten teams (between 50 and 125 people) that work together on one train. The special thing about this level is, that you do not work on projects, you work on so called programs. Huge companies normally have several trains and Programs. [26] Every requirement are administered by the Produt Manager on the program level and the Product Owner uses them to make a story for the different teams. After about five iterations, what would be around ten weeks, you create a program increment containing all the different increments that were implemented by the different teams. This outcome can, but must not be delivered by the Agile Release Train. [26]

The large solution is the third level. It is not necessary but optional and is mainly used when it comes to building large hard- or software systems. Its contains different elements like the solution train, solution intent or solution management.

The upper level is called the Portfolio Level. Here all the different programs are defined. Just like the Program Level, huge companies normally work with more than one portfolio and their teams. The portfolio is not only responsible for budget but also for target measures, while a normal investment is aimed between six and twelve month. [26] The target measures are called Epics and you differ between a Business Epic, which is customer oriented and an Enabler Epic, that is based on technical solutions. [26]

2.3 Patterns

2.3.1 Definition

"A pattern is a general, reusable solution to a common problem in a given context." [28] – Alexander M. Ernst, 2010

A pattern always consists of three different elements: a context, a problem and its solution. They are often used to communicate solutions for a common problem in various fields of software engineering, like a sample solution, that can be used over and over again. [29] Patterns are not obtained by developing them. To receive a pattern, you monitor how certain solutions are applied to a problem and document those results. But not every solution is a pattern. It needs to be applied at least three times to fall into this category. [30] This is what is called the rule of three: "[...] a good pattern should have three examples that show three insight-fully different implementations."

2.3.2 Pattern documentation

To describe a pattern normally a format, called the Alexandrian form is used. This form does not have a determined design, but shows a couple of possible elements for describing a pattern. Generally the first thing you do is to give this particular pattern that is being created a name. [31] After doing that you analyze the problem that has to be solved. That makes it easier for those, who want to use the pattern as well, because they'll know when exactly to apply it. After that you define the context, because it may only make sense to apply it in this specific context and not somewhere else. [31] This step increases the efficiency, making sure everyone else understands where it has to be used. You can also describe the forces, meaning the external influences that act upon the problem. Why is it so difficult to solve? It is not possible to affect the problem, but what can be done is to work against it, that's why a good pattern can resolve one, or even more forces. After analyzing the problem and trying to figure out why it is so difficult to solve, you may find a solution. To make the solution clear to everyone you do not only explain the solution, you describe the structure and behavior by describing how to build the solution, sometimes even showing different variants of it, because there might be a couple of different ways. It is also possible to draw a picture for most patterns, to visualize them and make the path of solving even clearer. Every author can extend the description of a patterns with more categories, like consequences, pros and cons using it, where else it was used, there is no particular schema that you have to stick to. [31]

Only a few patterns deserve to be called perfect and can stand on their own. A good pattern does not only find a solution to a problem, it also tells what forces might not have been solved or even if other patterns should be applied and how they changed the context. [31]

2.3.3 Pattern language

Different patterns can also be combined with each other, making it easier to compare them. The result is called Pattern Language. This is a sorted collection of several patterns that makes it easier for the user of this language to find a solution to his problem. This thesis is based on Large-Scale Agile Development Pattern Language (LSADPL) and can be describes with the following model (shown in figure 2.4 on page 17):

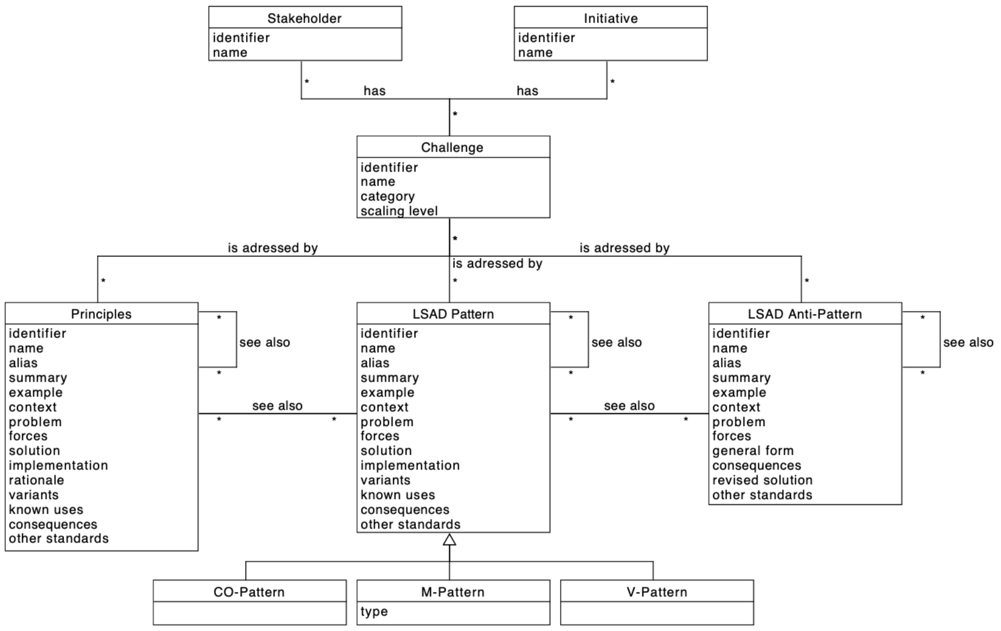


Figure 2.4: LSADPL UML Diagram [9]

This model categorizes patterns in five different categories:

- Coordination Patterns (CO-Patterns)
- Methodology Patterns (M-Patterns)
- Viewpoint Patterns (V-Patterns)
- Anti-Patterns (A-Patterns)
- Principles

Chapter 3

Related Work

In this chapter an overview of the current literature regarding the recurring concerns in Large-Scale Agile Development and the Pattern Language is given.

3.1 Related work on recurring concerns in large-scale agile development

This Thesis is part of the research project of Uludağ,Ö. and is thereby based on mainly two publications. In the publication "Identifying and Structuring Challenges in Large-Scale Agile Development based on a Structured Literature Review", Uludağ et al. are exploring the field of challenges in Large-Scale Agile Development by identifying and reviewing 73 relevant sources. As a result 79 challenges of 14 different stakeholders were being documented and categorized in 11 categories. [32]

Stakeholders:

- Agile Coach
- Business Analyst
- Development Team
- Enterprise Architect
- Portfolio Manager
- Product Manager

- Product Owner
- Program Manager
- Scrum Master
- Software Architect
- Solution Architect
- Support Engineer
- Test Team
- UX Expert

Categories:

- Culture & Mindset
- Communication and Coordination
- Enterprise Architecture
- Geographical Distribution
- Knowledge Management
- Methodology
- Project Management
- Quality Assurance
- Requirements Engineering
- Software Architecture
- Tooling

Based on this research, Uludağ et al. created the Large-Scale Agile Development Pattern Language, to address challenges and concerns in Large-Scale Agile Development because his literature review shows that there are many challenges, that are not being addressed within the literature. [9]

Another approach in identifying challenges of this field was made by Dikert et al. with their publication ”Challenges and success factors for large-scale agile transformations: A systematic literature review". [5] This literature review differs from Uludağ et al. in a way that Dikert et al. adding 29 success factors that have been identified and categorized.

Another paper worth mentioning was published by Julian M. Bass under the name "Artefacts and agile method tailoring in large-scale offshore software development programmes". [33] Bass conducted 46 practitioner interviews identifying 25 artifacts within 5 categories. Identifying concerns was not his primary goal but a necessity for his research.

3.2 Related work on pattern languages

Patterns and pattern languages were firstly introduced 1977 by Alexander et al. in the context of architectural planning for buildings and towns [34]. The North Face pattern, for instance, targets buildings which have a big side facing north. The sun will usually cast a long shadow behind them, so the pattern proposes to populate this area with things that do not require sunlight [35].

As mentioned in chapter 2, patterns are general solutions to common problems, while pattern languages are cohesive collections of patterns. Architectural planning, in contrast to software architecture, revolves around space and habitability. Nonetheless, pattern languages can help solving software architecture problems just as well [36].
In 1987, Cunningham and Beck first applied both patterns and pattern languages to software architecture. The authors created a pattern language to solve common problems in object-oriented programming [37]. According to Salingaros, patterns are "a powerful tool for controlling complex processes", explaining how patterns and pattern languages became successful in computer science [38].

Chapter 4

Identification of Recurring Concerns and Best Practices

The following chapter will show the findings on recurring concerns and best practices for Product Owners and Product Manager. Chapter four is therefor presenting the results, that the author gained in the interviews. First part of this chapter will be the description of the methodology, that was used for the interviews. After that all concerns, that where identified are listed. In the last part of this chapter the identified patterns, and the pattern candidates are being listed.

4.1 Methodology

The goal of this work was to identify recurring concerns and best practices in Large-Scale Agile Development, by analyzing problems found in recent literature and identify new concerns. For this we used semi-structured interviews as shown in the figure below.

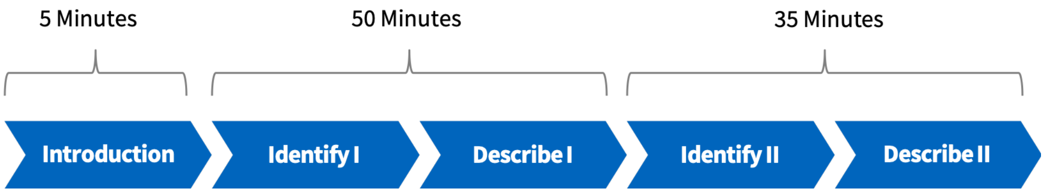


Figure 4.1: Structure of an interview

One interview was estimated for about 90 minutes and was divided into five different parts:

1. **Introduction:**
   The first five minutes are being used, that the participants can introduce themselves and tell something about their work and the company they work in.

2. **Identify I:**
   The participant was asked to identify at least three concerns that they face in their work as a Product Manager or Product Owner.

3. **Describe I:**
   Based on the identified concerns they are facing, they are describing the solutions, that are addressed to solve those concerns.

4. **Identify II:**
   In this phase the participants get a list of already existing concerns and are asked to tick those of, that they have been facing too.

5. **Describe II:**
   After going through that list they are ask to pick those three, that they have been facing the most and describe how they deal with those concerns in their daily work.

After finishing the interviews, the patterns, anti-patterns and if necessary new concerns are documented and analyzed. The outcome was then being send back to the participants to make sure that everything has been documented correctly and to get some feedback. If a pattern candidate was observed three or more times we consolidated the containing information and defined it as a pattern. This pattern is then correlated to existing patterns to bring it into the structure of the Large-Scale Agile Development Pattern Language.

4.2 Findings on recurring concerns

4.2.1 Concerns identified by Product Owner and Product Manager

As mentioned before, the first part of the interview was, that every participant should describe their most recurring concerns, that they have to deal with in their daily work life. Next they were asked to describe them to the author, so that they could be documented properly. Due to the limited time of the interview, most of them had time to explain three concerns, some of them described even more. After eleven interviews with seven Product

Owners and 4 Product Managers there were 23 new concerns discovered. Following all mentioned concerns are listed and described:

ID	Role	Interviewees Experience	Company's Experience	Industry	Company Size
1	Product Owner	3 - 6 years	More than 6 years	Service Sector	More than 200.000
2	Product Manager	1 - 3 years	1 - 3 years	Telecommunications	5001 - 10.000
3	Product Manager	1 - 3 years	1 - 3 years	Service Sector	11 - 50
4	Product Owner	3 - 6 years	More than 6 years	Service Sector	More than 200.000
5	Product Manager	1 - 3 years	More than 6 years	Financial Services, Insurance, Retail	100.001 - 200.000
6	Product Owner	3 - 6 years	More than 6 years	Service Sector	251 - 500
7	Product Owner	1 - 3 years	1 - 3 years	Transport, Logistics	More than 200.000
8	Product Owner	1 - 3 years	1 - 3 years	IT, Technology	1 - 10
9	Product Manager	1 - 3 years	3 - 6 years	Automotive	More than 200.000
10	Product Owner	1 - 3 years	3 - 6 years	Automotive	More than 200.000
11	Product Owner	1 - 3 years	3 - 6 years	Automotive	More than 200.000

- **C-101 Communication channels from higher hierarchy to lower:** The higher a hierarchy, the longer the communication channels. An information from a higher management level reaching to a low operational level can take time and information can get lost on the way. This concern is categorized as *Communication & Coordination* on the *Portfolio Level.*

- **C-102 Comparability of Storypoints outside teams/projects:** Storypoints are fictive and only make sense within one development team. Nevertheless, they're an important indicator for the complexity of a Story/Sprint and the productivity among team members. Thereby the possibility for further calculation and planning appears to be interesting. This concern is categorized as *Culture & Mindset* on the *Team*

Level.

- **C-103 Collision of Program Management and agile methods:** Usually a program contains several milestones, so deadlines that has to be met. But on the basis of agile principles, there will be no commitment to a distinct deadline months ahead. This concern is categorized as *Project Management* on the *Program Level.*

- **C-104 Stagnating continuous improvement process:** If a developer is in a long time project with mainly the same team, going through the continuous improvement process for too often, the actual improvement will stagnate because of missing new influences. This concern is categorized as *Methodology* on the *Team Level.*

- **C-105 Management dealing with a loss of control:** Managers have a steering role, controlling what the employees have to do and how to do it. Now due to agile practices, teams are self-organized and decide on their own how to reach a certain goal. This can feel like a loss of control to the management. This concern is categorized as *Culture & Mindset* on the *IT Organization Level.*

- **C-107 Spread agile mindset through entire organization:** Whether it's a project or a department within a company that works with agile methods, a lack of understanding of agile methods within other parts of the company will lead to communication gaps and incompatibilities to other stakeholders. This concern is categorized as *Culture & Mindset* on the *Enterprise Level.*

- **C-108 Identifying dependencies between teams:** When describing new tasks or User Stories in the Backlog, it might be that these tasks have dependencies between each other. These dependencies should be identified as early as possible, especially to reduce dependencies between several teams when assigning the tasks. This concern is categorized as *Communication & Coordination* on the *Program Level.*

- **C-109 Alignment of self-organized teams:** Agile teams organize themselves but also have to align in the big picture with the other teams. This rising complexity will result in an additional administrative burden to the Product Owner. This concern is categorized as *Culture & Mindset* on the *Program Level.*

- **C-110 Patience during the Agile Transformation:**
  The time for an Agile Transformation highly depends on the complexity of the organization. Such an organizational change promises benefits that will not be visible right away and have to be waited for by the employees. This concern is categorized as *Culture & Mindset* on the *Portfolio Level.*

- **C-111 Managing dependencies between teams:**
  In software development there will always be technical dependencies among developers and their tasks. The scaling of agile development will in most cases lead to dependencies among teams as well. Those dependencies are more difficult because the communication among teams is slower than among developers within a team. This concern is categorized as *Communication & Coordination* on the *Program Level.*

- **C-112 Disconnect between corp. strategy and execution:**
  An agile transformation often starts on the operational level, where most agile practices are being executed. The higher management on the strategical level is often missing agile experiences which leads incompatibilities between these levels and different working practices. This concern is categorized as *Communication & Coordination* on the *Portfolio Level.*

- **C-113 Understanding the demand for the Agile Transformation:**
  People tend to refuse change or anything that's new, if they don't see any need for a change. Changing processes and changing the way of working will also change working habits. And changing habits requires way more energy from the human brain than staying in the same habits. So people will not see the need without an external trigger. This concern is categorized as *Culture & Mindset* on the *Enterprise Level.*

- **C-114 Maintaining equal quality among teams:**
  An equal quality of teams is important in administer and planning a project. There are two main reasons for an imbalance between teams: Stuffing the team with unequal experienced developers in the first place and a different impact of the continuous improvement process of each team. This concern is categorized as *Software Architecture* on the *Team Level.*

- **C-115 Stakeholders being faced with higher amount of meetings:**
  Within the traditional process, stakeholders are used to define and approve a set of requirements and then just answering on upcoming questions during the implementation phase. In agile development there is a planning phase within each cycle and stakeholders will constantly find meetings (like Sprint Planning or Retrospective) in their calendar throughout the entire project. This concern is categorized as *Communication & Coordination* on the *Enterprise Level.*

- **C-116 Estimation of complex demands/requests:**
  The more complex a demand, the more difficult and time-consuming the estimation for it. But how to estimate an agile program when the Backlog just starts with a few user stories and will be more filled throughout the project. This concern is categorized as *Requirements Engineering* on the *Portfolio Level.*

- **C-117 Emotional impact of Agile Transformations to employees:**
  A change on employee's processes, work environments or practices has a direct impact on their habits. A very common first reaction of the human brain to changing habits is rejection. But rational thinking will say to do what the boss says. Every individual reacts differently to this conflict and in the worst case might need additional support. This concern is categorized as *Culture & Mindset* on the *Enterprise Level.*

- **C-118 Establish a common vision of the product:**
  A more complex project means more people are involved and in order for the project to be successful, everyone needs to have a common understanding of the "big picture". From the customer, over the business analyst to the developers and tester. This concern is categorized as *Project Management* on the *Portfolio Level.*

- **C-119 Improving processes in agile organizations:**
  Once the agile transformation is complete, an organization still has to improve continuously and adapt to external circumstances. This improvement process should then also follow the agile principles. This concern is categorized as *Methodology* on the *IT Organization Level.*

- **C-120 Product Owner lacking technical understanding:**
  The Product Owner acts as an interface between stakeholders and the development teams. Furthermore he is the owner of the product backlog. In this function he needs to have a certain depth of technical

understanding to be able to prioritize or divide tasks properly. This concern is categorized as *Software Architecture* on the *Program Level.*

- **C-122 Knowledge exchange between teams:**
  One agile team is a close cooperating unit, usually located within the same room or offices, where a knowledge exchange happens naturally or during the several Dailies, Retrospectives, etc. A knowledge exchange between several teams is not given like that. This concern is categorized as *Knowledge Management* on the *Program Level.*

- **C-124 Managing recurring requirements efficiently:**
  When the same requirements are not managed by the same team it might happen, that something is developed from scratch even though it could have saved a lot of money communicating This concern is categorized as *Project Management* on the *Program Level.*

- **C-125 Dealing with market specific requirements due to local circumstances:**
  An international operating company will not be able to roll-out all of their solutions world wide. Local circumstances like legal requirements, geographical differences or just a different scale of the companies offices can require different solutions in different markets for the same task. This concern is categorized as *Geographical Distribution* on the *Enterprise Level.*

- **C-126 Growing size and complexity of the Backlog:**
  A large-scale development program includes many different features that has to be managed in the Backlog. But with this growing complexity it is difficult to keep the overview over all user stories and their priorities and dependencies. This concern is categorized as *Requirements Engineering* on the *IT Organization Level.*

There are different levels, on which concerns can appear (shown in figure 4.2 on page 28). Each of the identified concerns are listed to one category, but it is possible that they appear on more than one level. In this thesis, there are three levels, that appear the most. Seven of the identified concerns were found on the Program Level and on the Enterprise Level and the Portfolio Level there were found five each. In the interviews there were only three concerns, that where stated to the Team Level and two were found on the IT Organization Level. Product Owners and Product Managers normally work on the Program Level, why it isn't surprising, that most of the concerns appear on this level. The other two levels, most concerns could be listed to the

Enterprise Level and the Portfolio Level. The Portfolio Level is right above the Program Level. It is starts and coordinates all new programs and delves with the requirements definition. Problems on this level can occur within the requirements definition or between the establishment and the common vision of the product. The Enterprise Level is not only the highest strategic level, but also maintains the IT organization and other departments. A lot of the key stakeholders are allocated in different departments, that's why the communication can be really challenging for Product Managers and Product Owners.

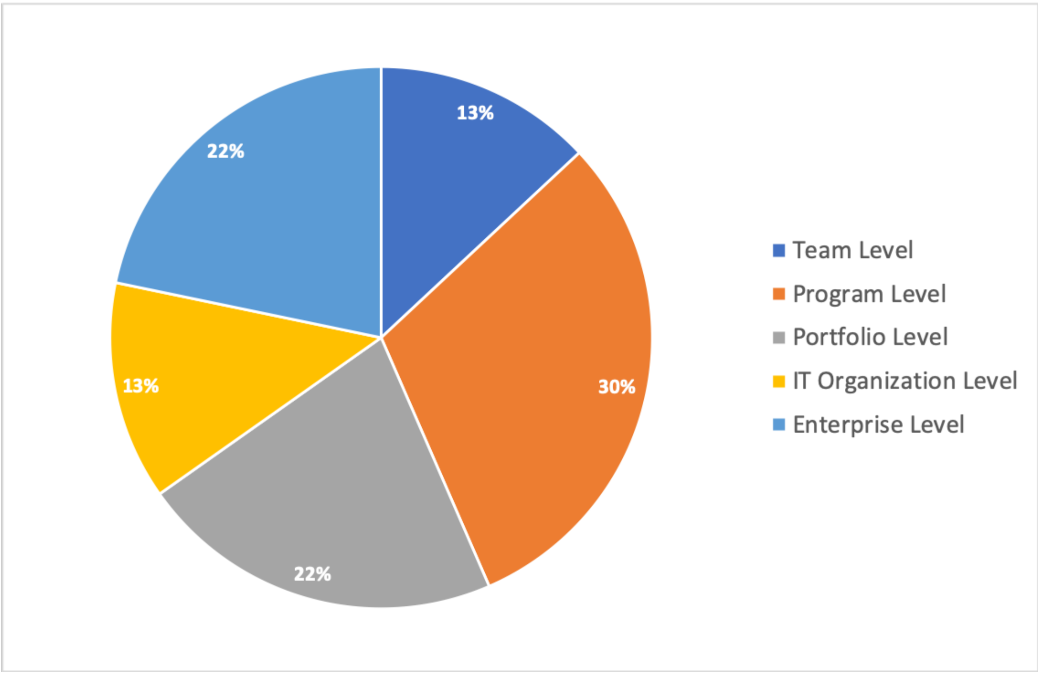


Figure 4.2: Identified concerns distributed in levels of scale.

Beside the different levels the concerns can be classified into, there are also different Categories (shown in figure 4.3 on page 29) which they can be divided into. All together there are eleven different categories. The identified concerns could be divided into eight of them, three of them (Tooling, Software Architecture and Quality Assurance) could not be detected. Most of the concerns appeared in Culture & Mindset and Communication & Coordination. The first one mainly focuses on the agile transformation and the change of view, every employee on all different hierarchical levels within the company has, changing into thinking in a agile way and not in the traditional anymore. Second, there are a lot of concerns listed within the category Communication

& Coordination. One of the main responsibilities of a Product Owner and a Product Manager is the control function and especially the Product Owner is a important interface between the Key Stakeholders and the Development Team. That's why it is not remarkable, that is one of the main categories for the observed concerns. The concerns delve into how to orchestrate the collaboration between the different teams in a complex project.

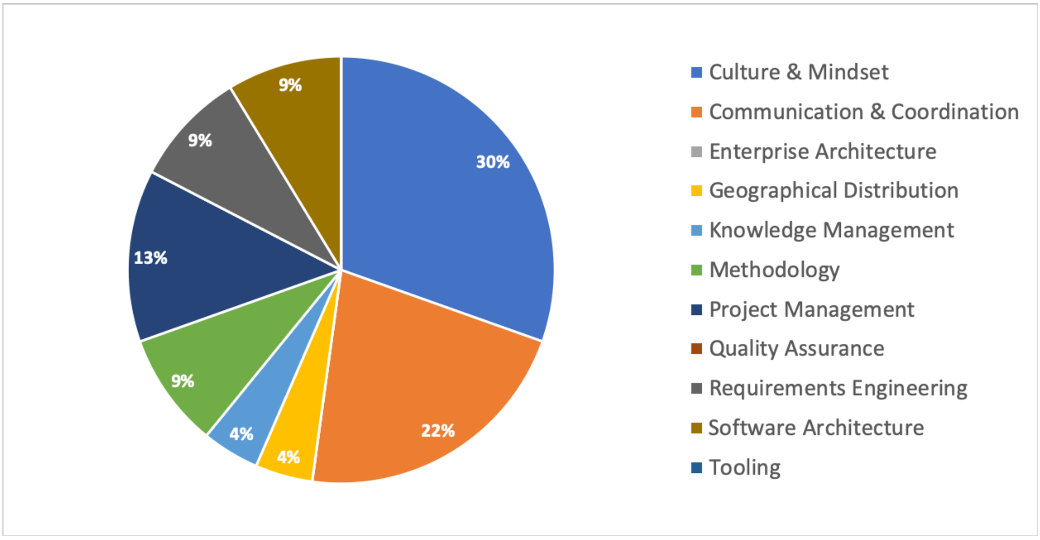


Figure 4.3: Identified concerns distributed in categories.

4.2.2 Identification of Recurring Concerns

In the second part of the interview every participant got a list of already identified concerns, including the ones found in the literature. They worked their way through the list to see, if any of those concerns recur in their company as well. The following figures show how many of the participants have the same problems and how important it is to find a sustainable solution for these concerns. All in all there were eleven Product Managers and Product managers interviewed, but not all of them got the same list of concerns, because some of them were identified later. Behind every concern there is a number, that shows how many of them got a list with this particular pattern on it. That is why "Growing size and complexity of the Backlog" and "Establishing a common understanding of agile thinking and practices" both have 100% even tough one of them was observed by eleven industry partners and the other one only by two.

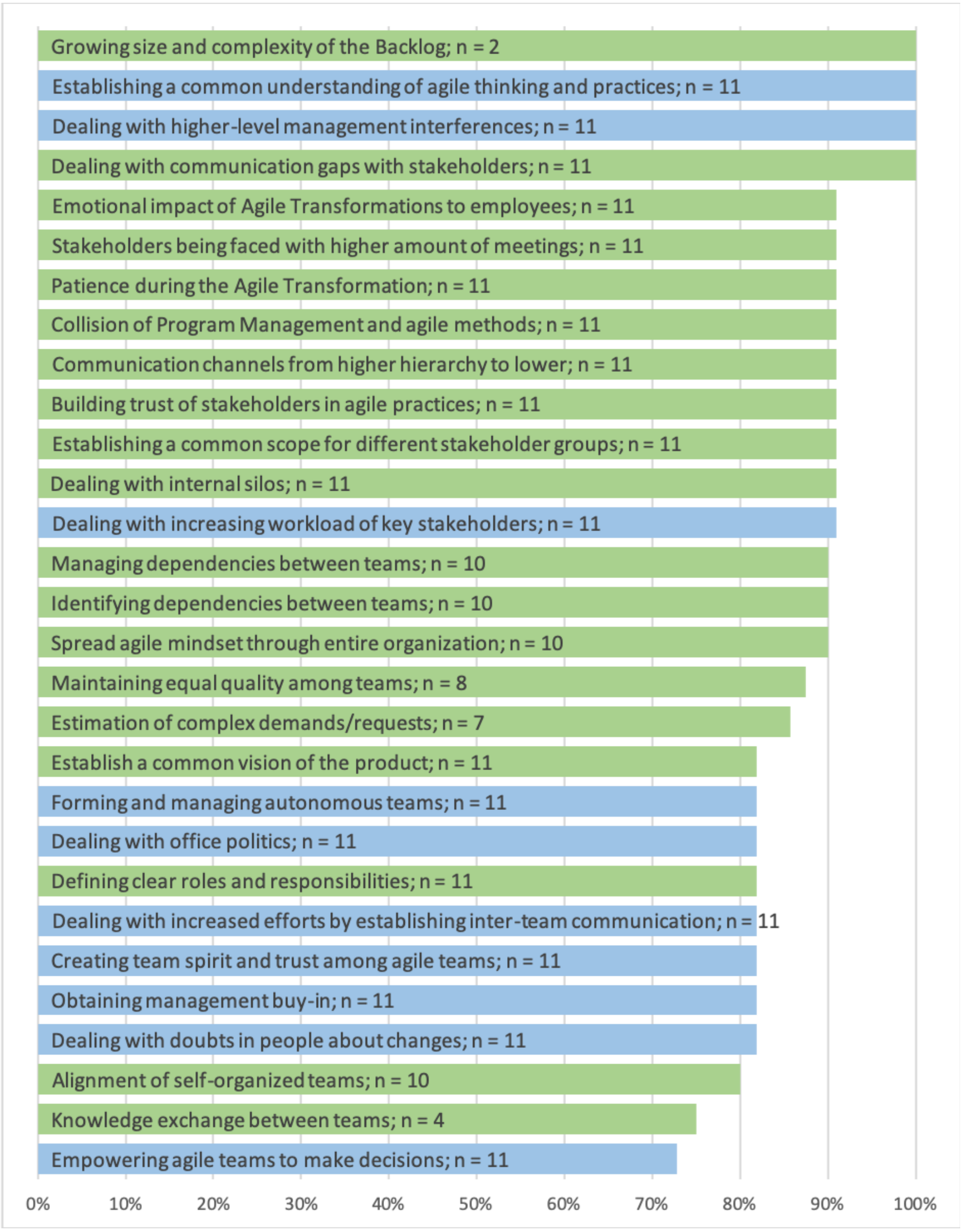


Figure 4.4: List of concerns by frequency of observation (Part 1)

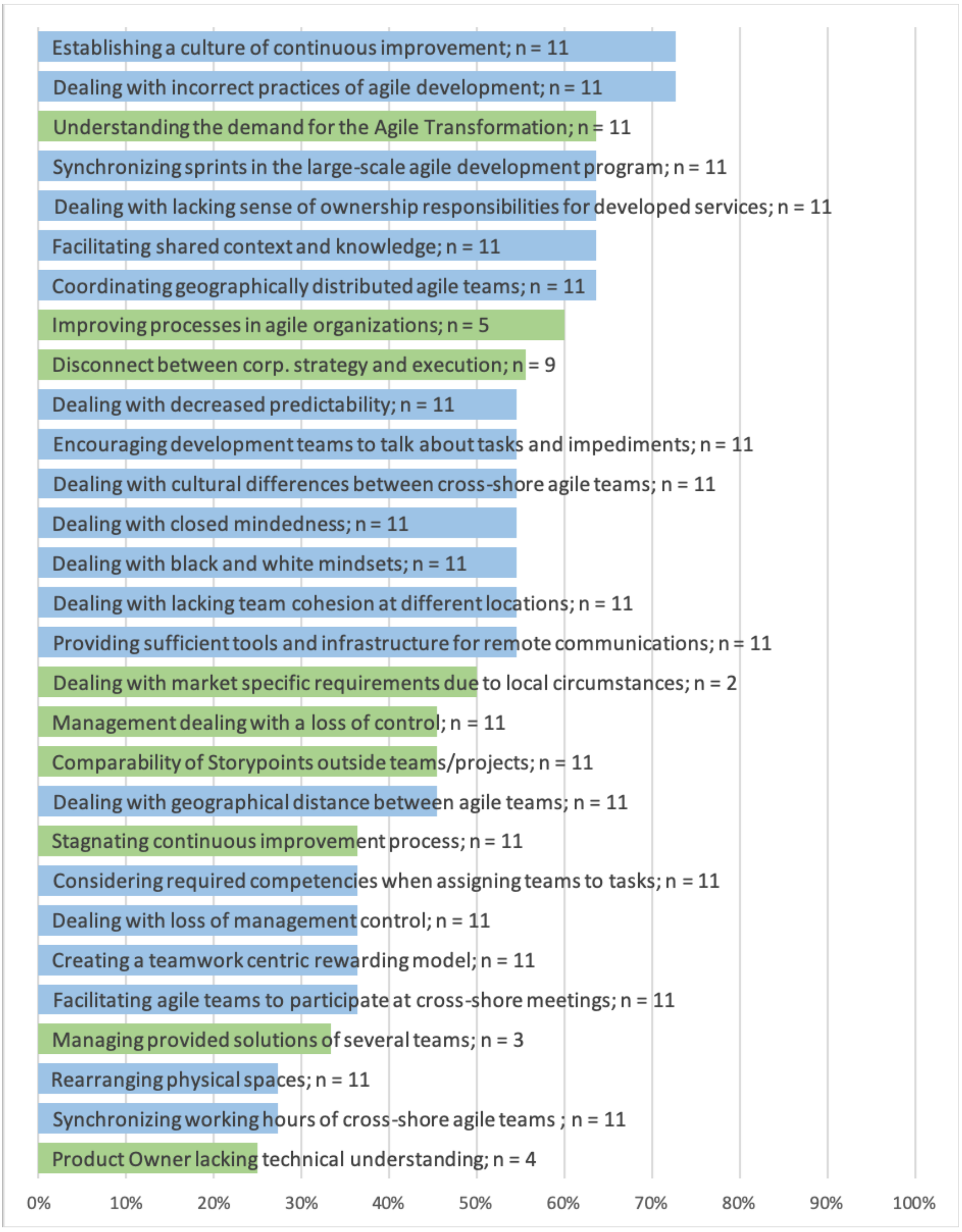


Figure 4.5: List of concerns by frequency of observation (Part 2)

4.3 Findings on Patterns

During the interview 36 pattern candidates have been identified. They got divided into 2 V-Patterns, 6 Principles, 17 M-Patterns, 8 CO-Patterns and 5 Anti-Patterns. As mentioned before, a pattern candidate turns into a pattern, when it is observed three times or more. The following figure shows the distribution of the different patterns. (shown in figure 4.6 on page 32)

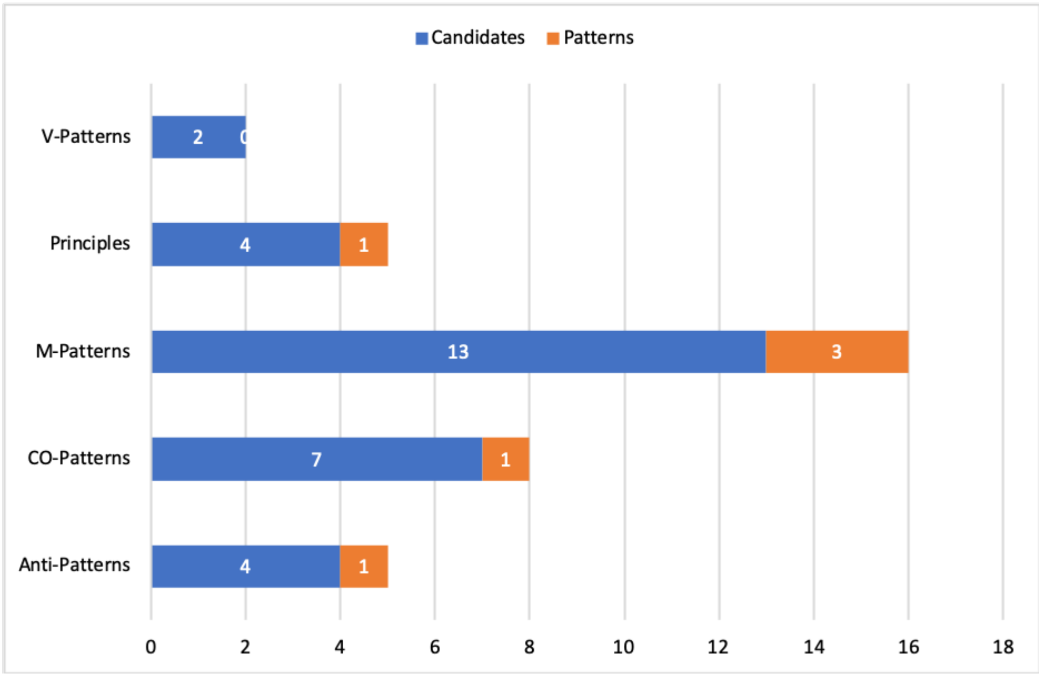


Figure 4.6: Identified patterns and pattern candidates by category

In the following all identified Patterns and Anti-Patterns are described:

- M-13 Magic estimation
- A-02 Don't think a change in too big steps
- CO-06 Program Increment (PI) Planning
- M-04 Domain Driven Design
- M-05 Feature Teams
- P-01 Fully transparent agile project

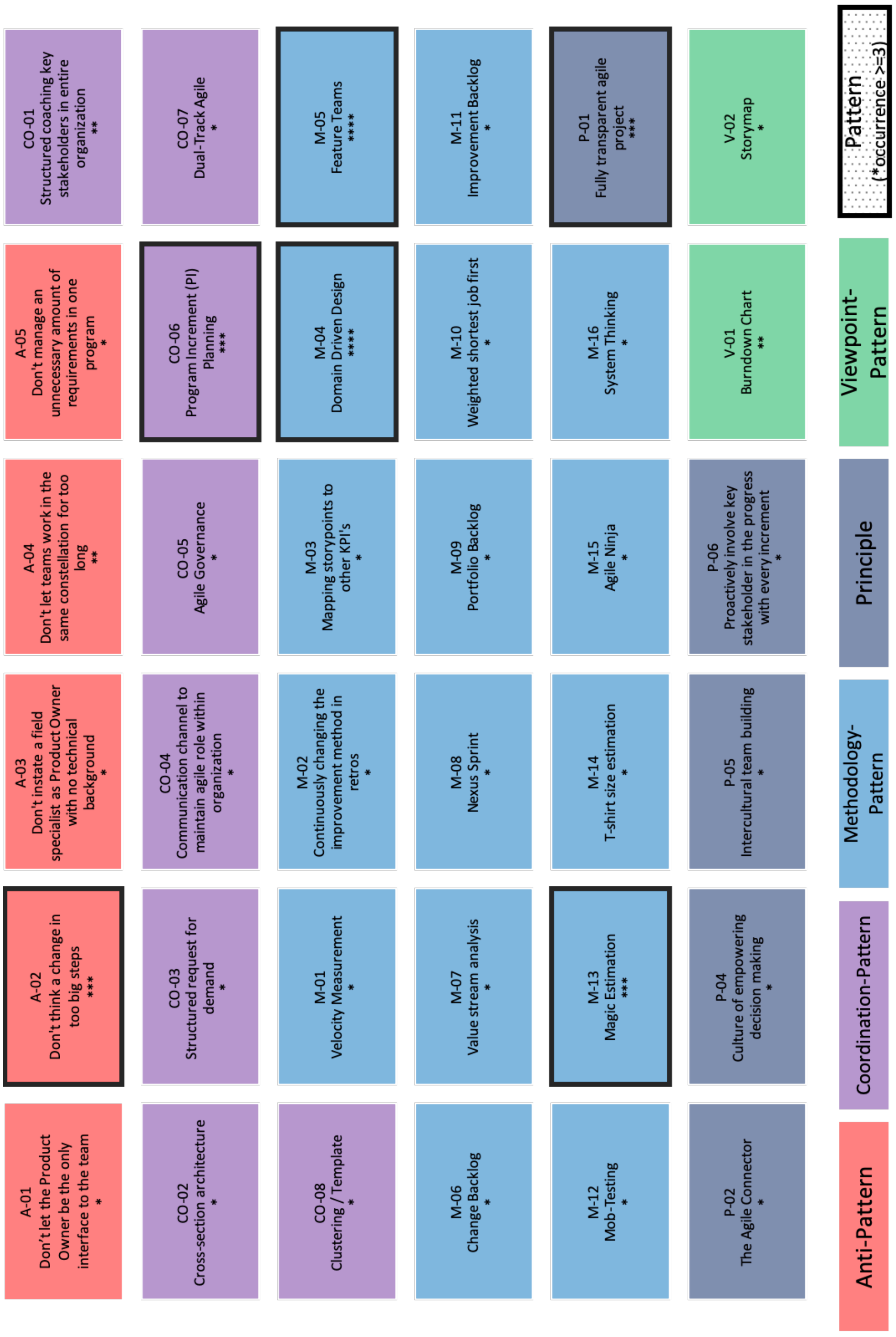

A-01 Don't let the Product Owner be the only interface to the team *
A-02 Don't think a change in too big steps ***
A-03 Don't instate a field specialist as Product Owner with no technical background *
A-04 Don't let teams work in the same constellation for too long **
A-05 Don't manage an unnecessary amount of requirements in one program *
CO-01 Structured coaching key stakeholders in entire organization **
CO-02 Cross-section architecture *
CO-03 Structured request for demand *
CO-04 Communication channel to maintain agile role within organization *
CO-05 Agile Governance *
CO-06 Program Increment (PI) Planning ***
CO-07 Dual-Track Agile *
CO-08 Clustering / Template *
M-01 Velocity Measurement *
M-02 Continuously changing the improvement method in retros *
M-03 Mapping storypoints to other KPI's *
M-04 Domain Driven Design ****
M-05 Feature Teams ****
M-06 Change Backlog *
M-07 Value stream analysis *
M-08 Nexus Sprint *
M-09 Portfolio Backlog *
M-10 Weighted shortest job first *
M-11 Improvement Backlog *
M-12 Mob-Testing *
M-13 Magic Estimation ***
M-14 T-shirt size estimation *
M-15 Agile Ninja *
M-16 System Thinking *
P-01 Fully transparent agile project ***
P-02 The Agile Connector *
P-04 Culture of empowering decision making *
P-05 Intercultural team building *
P-06 Proactively involve key stakeholder in the progress with every increment *
V-01 Burndown Chart **
V-02 Storymap *
Anti-Pattern
Coordination-Pattern
Methodology-Pattern
Principle
Viewpoint-Pattern
Pattern (*occurrence >=3)

Figure 4.7: All identified patterns and pattern candidates

4.3.1 Magic estimation

Pattern overview	
ID	M-13
Name	Magic estimation
Alias	–
Summary	Magic estimate is an exercise for Scrum teams to do a rough estimation on a whole product backlog. It is very fast and therefor delivers very good results.

Example
The department for demand management at InsuranceAG noticed, that they spend a lot of time estimating new demands. A request for demand is raised whenever another department wants to start an IT supported project. But based on the estimated time and costs, many requests were withdrawn, which makes the work on a very detailed estimation useless.

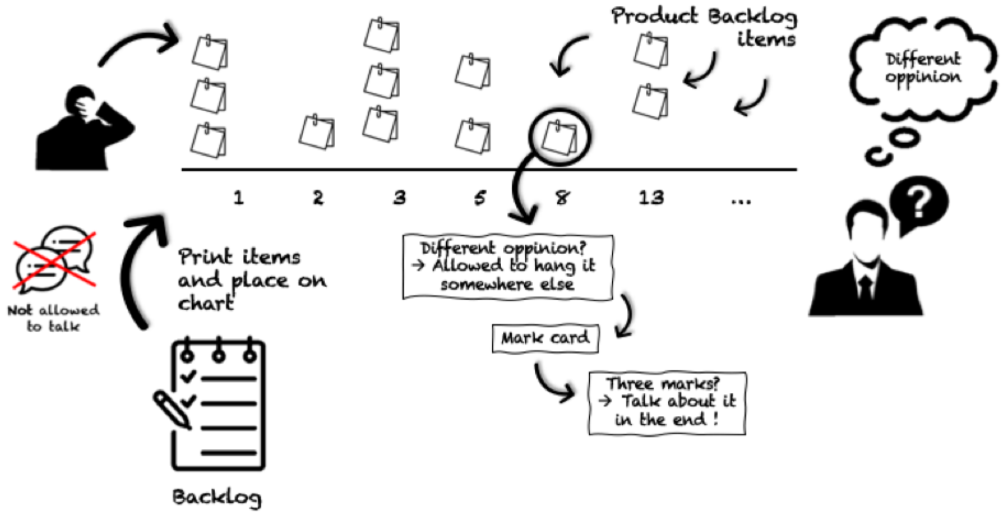
Context
Whenever there is a request for demand, it will not necessarily evolve into the start of a project.

Problem
The following concern is addressed by this M-Pattern:
C-116: Estimation of complex demands/requests

Forces
Sometimes a request for demand is placed to gain more information about the complexity about the project without the intention of starting it. Performing full requirements engineering is the perfect basis for a very concrete estimation, but too much effort when it's not clear whether the projects actually comes to fruition.

Solution
To perform the Magic Estimation a team of Requirements Engineers and experienced developers is needed. The Requirement Engineers have to roughly break down the demand into epics or User Stories, showing what most likely needs to be done to fulfill the demand. Based on this set, the experienced developers will now do a relative estimation, defining which Epic requires more or less effort than another one. They will start by defining a reference Epic with a known effort and give it a fictive number (for instance story points). The estimation will be based on the Fibonacci sequence

[1,2,3,5,8,13,21,34,55,89,144, ...], meaning every estimated item will be on this very scale and cannot be placed in between numbers. The developers will take turns placing a new item on the scale or move an existing one if they're not satisfied with its position. Items that are being moved too often will be taking out and be discussed at the end.


Product Backlog items
Different oppinion
1
2
3
5
8
13
...
Not allowed to talk
Print items and place on chart
Different oppinion? → Allowed to hang it somewhere else
Mark card
Three marks? → Talk about it in the end !
Backlog

Variants
Instead of the Fibonacci sequence, other fictive scales like the Cohn scale [0,1,2,3,5,8,13,20,40,100] can also be used. The procedure stays the same.

Consequences
Benefits:

- Easy to give a meaningful estimation in a short period of time
- Deduce when all tasks in the Backlog will be done
- If some team members disagree, they can talk about the problems and solve them
- Less effort in analysis

Liabilities:

- Less precise prediction

See Also
...

4.3.2 Don't think a change in too big steps

Pattern overview	
ID	A-02
Name	Don't think a change in too big steps
Alias	–
Summary	Big changes in an organization can cause negative effects. The bigger the change, the more likely it is, that the employees, who will be affected most by the change, will offer resistance. It is better, to implement changes in small steps, to make them sustainable.

Example
After identifying silos in a company and trying to break them open, they defined a goal that they wanted to achieve. They decided to adapt the change immediately with only one small interim solution, meaning that they applied a huge change in a short time.

Context
The problem appears, whenever the management of a company tries to counteract against internal silos with adapting change too quick and it is not accepted by the employees.

Problem
The following concern is addressed by this Anti-Pattern:
C-19: Dealing with internal silos

Forces
When planning a change, it sometimes appears difficult to break it down into smaller steps. The bigger the applied change, the higher the chance for employees offering resistance to it.

General Form
Implementing a change in the daily operations of a company can be challenging. Changing a running system isn't something that can be done overnight. Transform a system and develop a new way of doing things need time and a lot of effort. Choosing a new way of action and implement it without making adjustments or listen to what the employees want will not make all problems go away, it will rather cause even more problems. Change a system and make it work needs a lot of effort and interim steps, so that the employees can get used to the new way of doing things and to make adjustments if necessary.

It can even take several years to change a system properly and make it work.

Consequences
Benefits:

- Fast change/transformation

Liabilities:

- Risks the success of a change/transformation
- Increasing risk for resistance offered by the employees
- Current situation of the organization can get worse instead of better

Revised Solution
By downing the change into smaller steps and adapting the changes in smaller cycles.

See Also

- **M-06:** Change Backlog

4.3.3 Program Increment (PI) Planning

Pattern overview	
ID	CO-06
Name	Program Increment (PI) Planning
Alias	High-Level sprint planning
Summary	The Program Increment Planning is an event attended by several stakeholders to establish a common vision of the product, plan ahead the program by defining milestones and to identify dependencies among user stories and teams.

Example
At ConsultingCo they pried open all Epics into different User Stories and every team got an area of responsibilities. After merging the solutions of several teams, the outcome wasn't always as expected, because the individual teams didn't have the context that they would have needed.

Context
The more teams work on the same project, the more likely this problem occurs.

Problem
The following concerns are addressed by this CO-Pattern:
C-118: Establish a common vision of the product
C-126: Size and complexity of the backlog

Forces
It is difficult to fully communicate the product vision to every member of each team, but in many situations, it is important for the developer to understand the full context of a project. Large Scale Projects usually contain a lot of Epics, User Stories and Tasks, but the more items the backlog contains, the more difficult it is to manage and maintain it.

Solution
A Program Increment (PI) is a periodically created Increment, based on the Increments of the different teams, which can but must not be delivered and deployed. A PI usually take 8-12 weeks and always starts with the PI Planning.

"Program Increment (PI) Planning is a cadence-based, face-to-face [...], aligning all the teams [...] to a shared mission and Vision." Copyright © Scaled Agile, Inc.

This event will be attended by Product Owners, Product Managers, Business Stakeholders, Development Teams, Enterprise Architects and Requirement Engineers. Based on the most important and already estimated features in the backlog the attendees establish a common vision of the upcoming PIs. Thereby they will discuss the features more in detail, identify dependencies between features and between teams, define milestones and socialize within the entire program to build a stronger team spirit.

Variants
Depending on the scope and the complexity, not everyone has to attend the PI Planning. Sometimes it is enough if the Product Owner invites one representative from each agile team and business stakeholders in addition to that, if needed.

Consequences
Benefits:

- Establish a common vision of the product across all team members and stakeholders
- Identifying dependencies and document them
- Gaining a road map for the upcoming weeks and months

Liabilities:

- High planning effort

Other Standards
This pattern is also advised by the scaled agile framework (SAFe 4.6).

4.3.4 Domain Driven Design

Pattern overview	
ID	M-04
Name	Domain Driven Design
Alias	–
Summary	Domain Driven Design (DDD) helps conceptualize a development project while having a close communication to the concerned business experts. By dividing the software into domains, it will reduce the amount of dependencies

Example
At AutomotiveCo different departments were querying several requests and were constantly facing communication problems. The developers were missing on specific expertise which was important for the requested implementation. The problem was that with each request the department was collaborating with a different development team and always had to explain their field of expertise, either in whole or in part again.

Context
The problem occurs whenever a department is querying several requests and therefor collaborates with not the same but different development teams.

Problem
The following concerns are addressed by this M-Pattern:
C-111: Managing dependencies between teams
C-72: Considering required competencies when assigning teams to tasks
C-126: Growing size and complexity of the Backlog

Forces
Querying several requests to the IT organization does not mean, that the same developers implement all requests. Developers cannot be a specialist in any field possible and learning an additional field of specialty takes time.

Solution
Domain driven design describes domains of specialty, based on business departments, and allocates responsibilities within the IT Organization to main departments, departments or groups of development teams. These defined domains will gain specialty knowledge within that very business field and can thereby understand a new request more easily. The responsible developers or

departments will handle any current maintenance, any operations and any new request within this domain.

Variants
If the complexity of the domain requires it, it is possible to finer divide a domain into sub-domains. This can be helpful if a sub-domain is complex enough on its own but between the sub-domains within one domain may exist defined dependencies. Such known dependencies can be documented and then be handled faster or even be automated with a defined interface to it.

Consequences
Benefits:

- Stronger focus on the customer and his needs
- Developers gain a basic understanding of the customers business processes and will become experts in that field
- Developers will adopt business terms, which will lead to a common language
- Domains usually have little dependencies to other domains
- Clearly defined responsibilities
- The Backlog within a domain will not vary so much in size

Liabilities:

- Requires a certain size of the IT organization, otherwise the workload to the domains might not be distributed evenly
- If a domain grows out of its scope, the responsibilities to it have to be redefined

See Also
...

4.3.5 Feature Teams

Pattern overview	
ID	M-05
Name	Feature Teams
Alias	–
Summary	While planning a software project, the software will be divided into it's features. One feature will be implemented by one team. This will help reduce dependencies among teams.

Example
ConsultingCO had a program with three different applications and defined three different teams. One team per application. Since they were working agile, their User Stories in the backlog contain end-to-end functionality. Usually these functionalities had to be implemented on at least two applications. This led to technical dependencies between the teams and thereby additional effort in communication and planning.

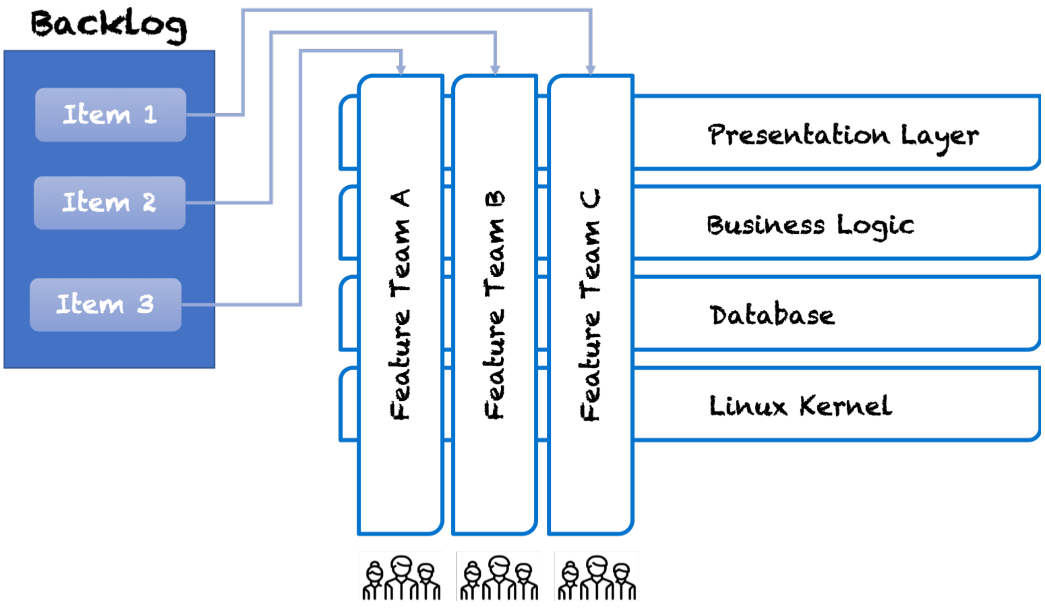
Context
This problem occurs whenever assigning tasks to different teams and is most likely to happen when structuring the program in traditional component teams.

Problem
The following concerns are addressed by this M-Pattern:
C-111: Managing dependencies between teams
C-72: Considering required competencies when assigning teams to tasks

Forces
A User Story should always contain one end-to-end functionality which can require development effort within different components, applications or systems. One end-to-end functionality should always be implemented within one sprint which might require for complex functionalities that more than one team has to work on it. One end-to-end functionality often requires implementation on different application layers which thereby needs several different specialists.

Solution
To avoid dependencies between the different teams it is necessary to implement feature teams. Feature teams are cross-functional and long-lived. They

choose their tasks one by one from the Product Backlog and complete them. Therefor they focus on the system productivity, rather than the productivity of a single component. The feature team has the complete responsibility for this task and each component of it. In order to be able to implement a feature on all application layers necessary, every team has developers with different skills. So a feature team could i.e. exist of four frontend developer, two backend developer and a database specialist.



Consequences
Benefits:

- Shared responsibilities
- Customer-centric approach
- Focus on system productivity instead of individual productivity
- Reducing dependencies between teams

Liabilities:

- Difficult to implement
- Need expert engineering practices

Other Standards
This pattern is also advised by Large-Scale Scrum (LeSS).

4.3.6 Fully transparent agile project

Pattern overview	
ID	P-01
Name	Fully transparent agile project
Alias	–
Summary	To actually understand agile methods, it's best to work/live it. The top level management usually lacks agile experience and understanding. Therefore the team needs to show full transparency to get all stakeholders on board and show them the benefits of an agile project management.

Example
At a company, a large-scale development project was implemented but the key stakeholders to this project were not so familiar with agile practices, which lead to communicational problems. The stakeholders did not understand why this project was running differently and thereby why the interaction and communication with the project was different as well.

Context
The less experience the organization has with agile methods, the more this will be a concern. The first people to fully understand the agile mindset are people who directly work with it (Developer team, Scrum Master, Product Owner). To get this mindset through the who organization reaching every possible stakeholder will take time. Until than there will be a gap of understanding that needs to be filled.

Problem
The following concerns are addressed by this P-Pattern:
C-44: Dealing with communication gaps with stakeholders
C-110: Patience during the Agile Transformation
C-113: Understanding the demand for the Agile Transformation

Forces
It is literally not possible to train every employee of an organization in agile methods at the same time. There will be people closer to the topic with more experience than others. This is why this problem is hard to solve.

Solution
Build a fully transparent project structure. It must be possible for all stakeholders to see every activity of the project. Public task boards are one example to do that. All stakeholders must have the possibility to attend every meeting. This is especially in the beginning of the project important to align everybody and make sure to have the same understanding on how to proceed.
In order to achieve that, the project needs a single point of information where the key information is stored and maintained. Key information can be:

- Project description
- Contact persons
- Meetings
- Task board(s)
- Milestones
- Documentation about the product (wiki)
- Documentation about interfaces and dependencies

Variants
A fully transparent project is a passive solution where the stakeholders must get active to close the gap of understanding. If that is not happening (usually when the gap is too big for that) then the project needs to have active communication to align the stakeholders. This would usually be done by the Product Owner.

Consequences
Benefits:

- Alignment of all stakeholders involved
- Giving the possibility for more understanding without forcing additional tasks to the stakeholders

Liabilities:

- Full transparency might be of inconvenience for the development team. Here communication to the team is needed to show them the need for this method

See Also

-

Other Standards

-

Chapter 5

Discussion

In the first part of the following chapter the key findings of this master thesis are discussed and the results, presented in chapter 4, are reflected. The second part shows the limitations this work had to deal with.

5.1 Key Findings

The research objective of this thesis were the recurring concerns of Product Managers and Product Owners in Large-Scale Agile Development and observing best practices solving those problems. A common technique to document these solutions are patterns, in this case the goal was to document them as a part of the Large-Scale Agile Development Pattern Language.

Findings on recurring concerns
During the interviews with the industry partners 23 new concerns were observed and the frequency of all in all 58 concerns was analyzed. Concerns in Large-Scale Agile Development can be classified into eleven different categories. The newly observed concerns could be classified into eight of them. Interview participants, who are in a state of agile transformation mainly indicated patterns within the Culture & Mindset category. Those, who do development performance for other companies mainly struggle with concerns within the Communication & Coordination and the Project Management category. The majority of the industry partners could state more concerns than best practices. That's a sign, that there is a need for a research in this field.

Findings on best practice solutions documented as patterns
After stating their concerns, the interview partner explained, what best prac-

tices they would apply for this particular concern. A best practice is documented as a pattern candidate. Altogether 36 pattern candidates were documented. The following six candidates were mentioned by three or more people from different companies and thereby evolved into a pattern:

- M-13 Magic estimation
- A-02 Don't think a change in too big steps
- CO-06 Program Increment (PI) Planning
- M-04 Domain Driven Design
- M-05 Feature Teams
- P-01 Fully transparent agile project

The documentation of these best practices was very well received by all interview partners. A lot of them were familiar with the concept of patterns and think, that this research is really relevant, because they still have to face a lot of concerns within Large-Scale Agile Development due to the fact, that this concept was created for small teams.

5.2 Limitations

The main source for this thesis is semi-structured interviews. Beside the evaluation of the information, that were gained in the interviews, the author also has the responsibility to do a critical reflection on the used method. When it comes to the reliability of a qualitative research, there are three criteria, that have to be checked: confirmability, dependability and credibility.

Confirmability: By confirmability it means the objectivity of the researcher. It is important to be objective as a researcher, meaning it can be helpful to use knowledge, that has been collected before the interview, just like literature and empirical knowledge. It is important, that the outcome is only based on the interviewee and not on the personal preferences of the researcher. Writing a study report might help to make the research as transparent as possible. [39, p.19]

Dependability: The understanding of dependability, also referred as reliability, is that the study should have the same outcome, if it would be repeated within the same circumstances, using the same methods and participants. [40, p.2] This is especially relevant for research, that is in its early stages. But nevertheless, the field of Large-Scale Agile Development is fast-moving, which leads to the assumption, that repeating the same study in a year of two could lead to different results, even though the same methods are applied to the same participants. There are three reasons for that: the first reason is, that within the literature new papers are delivered on a high frequency, meaning that some patterns and concerns identified within the interviews are later already being covered. Second, the organization, or environment where the interview partner operates in, can change, making some identified patterns or concerns obsolete, or leading to new ones. The third point is that, the individual can develop its own knowledge and experience within this field, leading to new best practices.

Credibility: The credibility of a study, also called the internal validity, "refers to how well an experiment is done, especially whether it avoids confunding (more than one possible independent variable [cause] acting at the same time). The less chance for confounding in a study, the higher its internal validity is." [41] It is important to make sure, that every interview was hold in the same condition and every interviewee has the same context. [39, p.19] To ensure the credibility of this thesis, every participant go the same material, used for the interviews, as well as a short overview on the topic including the goal of the interview and the current state of research. The participants were asked to name their most current concerns without knowing which ones were already identified, which helped them to focus on their concerns and not just on the ones that were already identified. It is expected, that because of the limited time frame for every interview, as well as the limited number of interviews, there are more concerns and patterns that could have been identified.

Chapter 6

Conclusion

This chapter summarizes the results of this thesis in chapter 6.1 and gives some insight on future work in 6.2.

6.1 Summary

For several years agile methods have been proven and brought a lot of benefits for their users. Due to the fact, that these methods were created for small teams it has been difficult for bigger companies to apply those methods and benefit from them. Scaling up agile methods is leading to a series of new concerns. Those concerns have been debated in the latest literature, but there are hardly any best practices documented.

The intention of this thesis was to observe recurring concerns, that Product Owners and Product Managers are facing in Large-Scale Agile Development. Therefor the author interviewed eleven stakeholders to determine their best practices in solving those concerns. Those best practices where documented as a pattern candidate, or if observed at more than three different industry partners, as a pattern within the Large-Scale Agile Development Pattern Language. In order to achieve that, three Research Questions were developed and were answered within this thesis:

Research Question 1: What are recurring concerns of Product Managers and Product Owners in Large-Scale Agile Development?

To discover the concerns Product Managers and Product Owners in Large-Scale Agile Development are facing, they were asked to name their most recurring concerns, that they are dealing with in their daily work. Altogether seven Product Owners and four Product Managers were interviewed. The concerns, that were found within the different interviews, were listed in chapter 4. Overall 23 new concerns were identified.

Research Question 2: What are best practices for addressing recurring concerns of Product Managers and Product Owners in Large-Scale Agile Development?

Based on those interviews the best practices on addressing those concerns were documented. When observed more than three times those best practices evolved into a pattern. If a solution has only been used less than three times, it is handled as a pattern candidate. 36 new pattern candidates were identified. 6 of them evolved into a pattern. Three of them where M-Patterns, one a CO-Pattern, one a Anti-Pattern and one a Principe. The patterns were described in chapter 4, whereas the pattern candidates can be found in the Appendix.

Research Question 3: Which anti-patterns should Product Managers and Product Owners avoid in large-scale agile development?

Other than Patterns, that show a solution to a concern, Anti-Patterns are a typical mistake, that is made in Large-Scale Agile Development and show how the concern should not be solved. Just like pattern candidates, anti-patterns candidates have to be observed at least three times to be called an anti-pattern. Within the interviews the author discovered five anti-pattern candidates, one of them was observed three times, so it turned into an anti-pattern.

6.2 Future Work

As mentioned before, this thesis had a limited time frame of five month. Investigating the current concerns for a longer time would add value to the research and would help to find best practices to solve those concerns in Large-Scale Agile Development. The research objective were Product Managers and Product Owners, but it would be possible to extend this to other stakeholders, working with LSAD to expand the knowledge that was gained within this thesis. The pattern candidates have to be examined further to evolve more of them into a pattern.
Also, pattern-based research design defines, that a pattern has to be proven in the industry. The time frame did not allow this, why the next step of this

research should be to fulfill this step.

Appendix A

Interviews on Identification of Recurring Concerns and Patterns

Note: If you have large models, additional evaluation data like questionnaires or non summarized results, put them into the appendix.

1. **General Questions**

   (a) Do you see yourself as a Product Owner or a Product Manager?

   (b) How is your role designated in your company?

   (c) How long have you been working in scaled agile development?

   (d) How long has your company been working with scaled agile development?

   (e) Does your company work internationally?

   (f) Which sector does your company operate in?

   (g) How many employees does your company have?

   (h) Are we allowed to contact you again for further research? (For example test reading the analyzed patterns or further questions regarding the given answers.)

2. **Questions on Recurring Concerns**

   (a) What are the most recurring concerns you face in your role as a Product Manager/Product Owner?

   (b) On which scaling level have you observed these concerns?

   (c) How often have you observed these concerns?
   (d) In which of these categories would you classify these concerns?

3. **Questions on Best Practices for most recurring Concerns**
   (a) How do you tackle this concern?
      i. Is there a concrete project or product, in which this pattern was used?
      ii. When does this problem occur?
      iii. Which concern can be addressed with this pattern?
      iv. Why is this concern difficult to solve?
      v. How can this concern be solved?
      vi. Are there different variants to this pattern?
      vii. What pros and cons do occur using this pattern?
      viii. Is this pattern used in combination with any others?
      ix. What other standards or frameworks recommend this pattern?
      x. Where does the data for this pattern come from?
   (b) What should you avoid to not have this concern?
      i. Is there a concrete project or product, in which this anti-pattern was used?
      ii. When does this problem occur?
      iii. Which concern can be addressed with this anti-pattern?
      iv. Why is this concern difficult to solve?
      v. How do you find this anti-pattern in practice?
      vi. What pros and cons do occur using this anti-pattern?
      vii. How can this anti-pattern be avoided?
      viii. Is this anti-pattern used in combination with any other pattern?
      ix. What other standards or frameworks recommend this anti-pattern?

4. **Questions on Concerns identified in Literature**
   (a) Which of those concerns, that have been identified in the literature, have you observed in your role as Product Owner/Product Manager?
   (b) How do you tackle this concern?

i. Is there a concrete project or product, in which this pattern was used?
ii. When does this problem occur?
iii. Which concern can be addressed with this pattern?
iv. Why is this concern difficult to solve?
v. How can this concern be solved?
vi. Are there different variants to this pattern?
vii. What pros and cons do occur using this pattern?
viii. Is this pattern used in combination with any others?
ix. What other standards or frameworks recommend this pattern?
x. Where does the data for this pattern come from?

(c) What should you avoid to not have this concern?

i. Is there a concrete project or product, in which this anti-pattern was used?
ii. When does this problem occur?
iii. Which concern can be addressed with this anti-pattern?
iv. Why is this concern difficult to solve?
v. How do you find this anti-pattern in practice?
vi. What pros and cons do occur using this anti-pattern?
vii. How can this anti-pattern be avoided?
viii. Is this anti-pattern used in combination with any other pattern?
ix. What other standards or frameworks recommend this anti-pattern?

5. **Discussion**

(a) Do you have any additional comments?

Appendix B

Documentation of New Concerns

ID	Name	Description	Category	Scaling Level
C-101	Communication channels from higher hierarchy to lower	The higher a hierarchy, the longer the communication channels. An information from a higher management level reaching to a low operational level can take time and information can get lost on the way.	Communication & Coordination	Portfolio Level
C-102	Comparability of Storypoints outside teams/projects	Storypoints are fictive and only make sense within one development team. Nevertheless, they're an important indicator for the complexity of a story/sprint and the productivity among team members. Thereby the possibility for further calculation and planning appears to be interesting.	Culture & Mindset	Team Level
C-103	Collision of Program Management and agile methods	Usually a program contains several milestones, so deadlines that has to be met. But on the basis of agile principles, there will be no commitment to a distinct deadline months ahead.	Project Management	Program Level
C-104	Stagnating continuous improvement process	If a developer is in a long time project with mainly the same team, going through the continuous improvement process for too often, the actual improvement will stagnate because of missing new influences.	Methodology	Team Level
C-105	Management dealing with a loss of control	Managers have a steering role, controlling what the employees have to do and how to do it. Now due to agile practices, teams are self-organized and decide on their own how to reach a certain goal. This can feel like a loss of control to the management.	Culture & Mindset	IT Organization Level

ID	Name	Description	Category	Scaling Level
C-107	Spread agile mindset through entire organization	Whether it's a project or a department within a company that works with agile methods, a lack of understanding of agile methods within other parts of the company will lead to communication gaps and incompatibilities to other stakeholders.	Culture & Mindset	Enterprise Level
C-108	Identifying dependencies between teams	When describing new tasks or user stories in the Backlog, it might be that these tasks have dependencies between each other. These dependencies should be identified as early as possible, especially to reduce dependencies between several teams when assigning the tasks.	Communication & Coordination	Program Level
C-109	Alignment of self-organized teams	Agile teams organize themselves but also have to align in the big picture with the other teams. This rising complexity will result in an additional administrative burden to the Product Owner.	Culture & Mindset	Program Level
C-110	Patience during the Agile Transformation	The time for an agile transformation highly depends on the complexity of the organization. Such an organizational change promises benefits that will not be visible right away and have to be waited for by the employees.	Culture & Mindset	Portfolio Level
C-111	Managing dependencies between teams	In software development there will always be technical dependencies among developers and their tasks. The scaling of agile development will in most cases lead to dependencies among teams as well. Those dependencies are more difficult because the communication among teams is slower than among developers within a team.	Communication & Coordination	Program Level

ID	Name	Description	Category	Scaling Level
C-112	Disconnect between corp. strategy and execution	An agile transformation often starts on the operational level, where most agile practices are being executed. The higher management on the strategical level is often missing agile experiences which leads incompatibilities between these levels and different working practices.	Communication & Coordination	Portfolio Level
C-113	Understanding the demand for the Agile Transformation	People tend to refuse change or anything that's new, if they don't see any need for a change. Changing processes and changing the way of working will also change working habits. And changing habits requires way more energy from the human brain than staying in the same habits. So people will not see the need without an external trigger.	Culture & Mindset	Enterprise Level
C-114	Maintaining equal quality among teams	An equal quality of teams is important in administer and planning a project. There are two main reasons for an imbalance between teams: Stuffing the team with unequal experienced developers in the first place and a different impact of the continuous improvement process of each team.	Software Architecture	Team Level

ID	Name	Description	Category	Scaling Level
C-115	Stakeholders being faced with higher amount of meetings	Within the traditional process, stakeholders are used to define and approve a set of requirements and then just answering on upcoming questions during the implementation phase. In agile development there is a planning phase within each cycle and stakeholders will constantly find meetings (like sprint-planning or retro) in their calendar throughout the entire project.	Communication & Coordination	Enterprise Level
C-116	Estimation of complex demands/requests	The more complex a demand, the more difficult and time-consuming the estimation for it. But how to estimate an agile program when the backlog just starts with a few user stories and will be more filled throughout the project.	Requirements Engineering	Portfolio Level
C-117	Emotional impact of Agile Transformations to employees	A change on employee's processes, work environments or practices has a direct impact on their habits. A very common first reaction of the human brain to changing habits is rejection. But rational thinking will say to do what the boss says. Every individual reacts differently to this conflict and in the worst case might need additional support.	Culture & Mindset	Enterprise Level
C-118	Establish a common vision of the product	A more complex project means more people are involved and in order for the project to be successful, everyone needs to have a common understanding of the "big picture". From the customer, over the business analyst to the developers and tester.	Project Management	Portfolio Level

ID	Name	Description	Category	Scaling Level
C-119	Improving processes in agile organizations	Once the agile transformation is complete, an organization still has to improve continuously and adapt to external circumstances. This improvement process should then also follow the agile principles.	Methodology	IT Organization Level
C-120	Product Owner lacking technical understanding	The Product Owner acts as an interface between stakeholders and the development teams. Furthermore he is the owner of the product backlog. In this function he needs to have a certain depth of technical understanding to be able to prioritize or divide tasks properly.	Software Architecture	Program Level
C-122	Knowledge exchange between teams	One agile team is a close cooperating unit, usually located within the same room or offices, where a knowledge exchange happens naturally or during the several dailys, retros, etc. A knowledge exchange between several teams is not given like that.	Knowledge Management	Program Level
C-124	Managing recurring requirements efficiently	When the same requirements are not managed by the same team it might happen, that something is developed from scratch even though it could have saved a lot of money communicating	Project Management	Program Level
C-125	Dealing with market specific requirements due to local circumstances	An international operating company will not be able to roll-out all of their solutions world wide. Local circumstances like legal requirements, geographical differences or just a different scale of the companies offices can require different solutions in different markets for the same task.	Geographical Distribution	Enterprise Level

ID	Name	Description	Category	Scaling Level
C-126	Growing size and complexity of the Backlog	A large-scale development program includes many different features that has to be managed in the Backlog. But with this growing complexity it is difficult to keep the overview over all user stories and their priorities and dependencies.	Requirements Engineering	IT Organization Level

Appendix C

Documentation of Pattern Candidates

C.1 Documentation of Coordination Pattern Candidates

C.1.1 Structured coaching an entire organization

Pattern overview	
ID	CO-01
Name	Structured coaching key stakeholders in entire organization
Alias	–
Summary	In many cases just one or a few projects/departments work with agile methods but not the entire organization. Nevertheless there will be dependencies or interactions with other stakeholder. In order for such a collaboration to be effective, both sides should at least have a common understanding of agile methods. This is why it can be very usefull to also coach employees who don't necessarily work agile.

Example

At MobileCo one internal program initiated the need for agile practices. Being the first agile program a lot of communicational gaps towards their stakeholders existed. Since many processes where not aligned to agile practices the program could not gain all benefits.

Context
This problem occurs whenever an agile transformation of the organization is planned or ongoing. Furthermore this problem can occur, when having an agile project without the rest of the organization aligning to it.

Problem
The following concerns are addressed by this CO-Pattern:
C-110: Spread agile mindset in entire organization
C-86: Emotional Consequences of Agile Transformations
C-87: Patience during the Agile Transformation

Forces
It is literally not possible to train an entire organization in agile methods at the same time. It usually starts within one project or one department but then has to spread more into the organizational structure. But during this process will be a gap between employees already aligned and people not having been trained in agile methods yet. Not everyone has the same knowledge requirement regarding agile methods.

Solution
An agile transformation within a company usually starts with one project. This is project needs to be trained in agile methods first. If that agile approach was successful and should be used in more projects within the company the management buy-in has to be obtained. As soon as the goal is to be an agile organization the agile mindset has to be spread in the whole organization. To be overly productive and efficient, it is necessary to train employees as agile coaches. Furthermore, key positions, that have the highest need for a coaching, have to be identified. A common key position are Stakeholders, working together with an agile project.
An extended understanding of agile methods within the organization encourages the alignment of agile processes and the understanding and the need for an agile organization.

Variants
The coaching can exclusively be done by external coaches but by training internal employees as coaches who have a better understanding of the day-to-day business they are an efficient source that costs less money in the long run.

Consequences
Benefits:

- To align the entire organization is very sustainable and will be very useful in the long run
- Better alignment of agile processes
- Common understanding for the need of an agile transformation

Liabilities:

- This approach will take a lot of time. Depending on the organizational size even years.
- Not every department should necessarily work agile. There needs to be an evaluation where the coaching makes sense and where not.

See Also
C-Pattern: Fully transparent agile project

C.1.2 Coordination of current dependencies

Pattern overview	
ID	CO-02
Name	Coordination of current dependencies
Alias	–
Summary	In Large-Scale agile programs there will always occur dependencies between the teams. This pattern aims to identify dependencies early and manage them in a efficient way.

Example
At ConsultingCo an agile program was executed. Due to the complexity dependencies between the teams could not be eliminated. Some dependencies were identified late during the implementation process which made it difficult to manage them putting this sprint goal into risk.

Context
This problem might occur whenever several agile teams are working within the same project.

Problem
The following concerns are addressed by this CO-Pattern:
C-108: Identifying dependencies between teams
C-111: Managing dependencies between teams

Forces
Large-Scale agile programs gain a complexity where dependencies among teams cannot be excluded completely. The later dependencies are identified the more difficult it is to react on it.

Solution
An active communication and knowledge exchange between the teams will help identify dependencies early and manage current dependencies. The exchange between the teams can happens in three different ways: 1. Cross-section architecture: Providing a cross-section architecture, will bring more transparency to the project. Each team will understand their role within the big picture and will also understand what the other teams will provide. 2. Scrum Master Meeting: Every team plans their own Sprint. After that the Scrum Masters meet up and discuss their outcome, to identify dependencies and try to coordinate them. 3. Every Scrum Master attends every daily of

every team: The dailies of every team are coordinated at a different time, to make sure every Scrum Master can attend every Daily.

Variants
If the complexity of the projects requires an even closer collaboration between the teams, the Scrum of Scrum Meeting defines a stricter way of the steps described above.

Consequences
Benefits:

- Early identification of dependencies
- Constant coordination of dependencies
- Close knowledge exchange between teams
- Creating a team spirit not only within the team, but among all teams

Liabilities:

- High amount of coordinating meetings

See Also
-

C.1.3 Structured request for demand

Pattern overview	
ID	CO-03
Name	Structured request for demand
Alias	–
Summary	If a customer has a complex demand and requests an estimation, this can be very time consuming and in the end the customer might not go through with the project. A structured request for demand requires the customer to answer a set of questions in order to raise a demand. By answering these questions the customer will have to think about the business case and how this will bring him value. This can prevent unnecessary requests.

Example
At InsuranceCo the business department was sending a request for a new online platform. They asked how long it would take to apply this and how much it would cost. Due to the fact, that is was a bigger project, the IT department took about two weeks to estimate time and price. After the IT discussed everything with the business department they decided, that it would be to expensive. On this account it was a waste of effort for the IT.

Context
This problem occurs whenever requested demands have to be estimated before they evolve into a project.

Problem
The following concern is addressed by this CO-Pattern:
C-106: Estimation of complex demands/requests

Forces
Sometimes a request for demand is placed to gain more information about the complexity about the project without the intention of starting it. Performing full requirements engineering is the perfect basis for a very concrete estimation, but too much effort when it's not clear whether the projects actually comes to fruition.

Solution
The process for requesting a demand within an organization needs to de-

fine what kind of information are required for placing a request. Thereby a request of demand has to contain information about:

- Business value
- Related business units
- Additional positive impact like saving of time
- Effort/Impact Analysis

By defining this information as required it ensures that the stakeholders have to make sure, their demand is actually business relevant and this lowers the risk of unnecessary workload on estimation.

Variants
The required information for a request of demand can vary, based on several factors like company size or industry.

Consequences
Benefits:

- Less unnecessary requests
- The quality of a request is ensured, leading to a more efficient process

Liabilities:

- Increased time exposure for the business departments

See Also
M-13: Magic Estimation

C.1.4 Communication channel to maintain agile role within organization

Pattern overview	
ID	CO-04
Name	Communication channel to maintain agile role within organization
Alias	–
Summary	Sometimes people cannot fully put their agile role into practice because their higher management doesn't work with agile methods and thereby setting them borders. An additional comm. channel can help contest a decision from out of the agile environment if necessary.

Example
A Product Owner was running a program within his company, managing the Backlog. According to agile practices the PO is the only person responsible for managing the Backlog. In this company the program was agile because it was an agile IT organization, but the rest of the company was not agile. Due to high hierarchies, the PO got overruled in his prioritization, meaning he couldn't fully live his agile role.

Context
This problem occurs whenever the operational level is working agile but the processes on the strategy level are not aligned.

Problem
The following concern is addressed by this CO-Pattern:
C-112: Disconnect between corp. strategy and execution

Forces
If an IT organization decides to work completely agile, it still has no authority above that. Agile principles are not considering high hierarchies.

Solution
One should always try to maintain his agile role with all entitlements and duties. If an organizational unit (e.g. an IT organization) is becoming agile but the structure around it still isn't, there has to be communication channels that help maintaining one's agile role.
If an agile role gets restricted in its doing, it needs to communicate that

disconnect to the corresponding manager on the strategic level. This can be a structured workflow in form of a request. That request needs to contain the following information:

- Related strategical decision
- Content of that decision
- Requested deviation to the decision
- Reason for the deviation
- Benefits of changing decision
- Related agile value/principle/practice

The potential for a successful request increases if the communication channel is not direct from any individual to a strategic manager directly, but through a "trusted middleman". That middleman would be a high management position within the agile organizational unit (e.g. strategic management of IT organization).

Variants
The description of the communication channel up the hierarchy, cannot be described in detail because it'll vary from organization to organization.

Consequences
Benefits:

- Better alignment of agile and non-agile organizational units
- Attempt to stick to the framework as much as possible
- Helps to close communication gaps
- Helps to understand the need for becoming agile

Liabilities:

- No guarantee for success
- Requires management buy-in

See Also
-

C.1.5 Agile Governance

Pattern overview	
ID	CO-05
Name	Agile Governance
Alias	–
Summary	An agile organization can apply a continuous improvement process using agile methods as well. A regular meeting for improvement proposals and an immediate application of a solution after a successful decision making process can bring the same speed from agile projects to agile organizations.

Example
At LogisticsCo, a department was driving an ongoing agile transformation. During this process, concerns occurred that had to be addressed towards their higher management.

Context
This concern occurs within every agile organization of a large scale.

Problem
The following concern is addressed by this CO-Pattern:
C-119: Improving processes in agile organizations

Forces
Applying changes to a large organization always contains the risk of being unsuccessful or leading to some kind of unplanned disadvantage. But organizational changes often appear in a long cycle time.

Solution
Employees in an agile environment should have an opportunity to scrutinize the development of the organization and how to improve processes for a better agile alignment. The Agile Governance Meeting is an event that is taking place every three month. Employees can suggest topics about an organizational change that shall be part of the agenda and be discussed. Two employees are responsible for planning this event and afterwards the responsibility will switch. Anyone is invited to attend this meeting, but it is not mandatory and the attendance for each person highly depends on the topics on the agenda.

When a topic from the agenda is discussed, it follows a distinct scheme to keep it efficient:

- A proposal will be created
- Questions for clarification can be raised
- A discussion round is established
- The proposal will be reworked based on the discussion
- Possible objections can be made
- Counter question the proposal. “What are possible risks?”
- Vote for or against the proposal
- If a proposal got approved, it effective immediately if not defined differently

Variants
The scheme for the Agile Governance Meeting can vary, depending on different factors within the company.

Consequences
Benefits:

- Scrutinizing the organization and the agile transformation
- Continuous improvement on an organizational level

Liabilities:

- Since the participation is voluntary, it cannot be assured that all relevant people are reached

See Also
-

C.1.6 Dual-Track Agile

Pattern overview	
ID	CO-07
Name	Dual-Track Agile
Alias	–
Summary	Dual-Track Agile is a method, that assures that the approach solving a problem always is the most efficient and best. This is enabled by a Dual-Track System, meaning there is not only a Delivery Track, but also a Discovery Track.

Example
At DevelopmentCo, a program was very well planned and implemented. After finishing the User Story and presenting it to the other teams, a other team told them, that there was a library implemented shortly before, that would have solved this problem even better.

Context
The problem occurs, whenever the different teams don't communicate together properly, which entails that there is no knowledge exchange.

Problem
The following concern is addressed by this CO-Pattern:
C-121: Knowledge exchange between teams

Forces
Each agile team is a self-organizing unit, who's strong collaboration will automatically lead to knowledge exchange within a team. But there is no defined process for an exchange between several teams. If the work, that the different teams are doing, isn't transparent enough, the other teams might not know, that they can actually benefit from each other.

Solution
To solve this concern Dual-Track Agile was developed. Dual-Track means, that a team works on two parallel tracks: Discovery Track and Delivery Track. Those two sub-areas work hand-in-hand. The outcome of the Delivery Track, where Product Owner and Usability Engineer work on together are for example new concepts or prototypes, still a mere hypothesis. The Usability Engineer and the Product Owner are responsible for gathering information and the concept.

The Development Track contains the actual development of all planned features. In this whole development process, they support each other by testing the outcome and improve it. Even if something from the Discovery track is discarded, it can still add value to their future work. It's important, that the whole team knows its responsibility for the project and that everyone has to take part in both tasks.

Variants
-

Consequences
Benefits:

- Continuous improvement
- Mutual support within the team

Liabilities:

- Developing an idea does not implement, that it is actually profitable, meaning it might be a waste of time
- Finding a new best practice does not automatically mean, that it is worth the effort

See Also
-

C.1.7 Clustering / Template

Pattern overview	
ID	CO-08
Name	Clustering / Template
Alias	–
Summary	Standardizing the different parameters of similar projects by analyzing their goal, input, outcome and their implementation helps to save work capacity and money.

Example
At AutomotiveCo they always started planning new projects from scratch, even though they could have used their know-how from the previous projects. Applying their knowledge from previous projects could have saved them a lot of time and money.

Context
This problem occurs when different teams and suppliers that are responsible for the projects.

Problem
The following concern is addressed by this CO-Pattern:
C-124: Managing recurring requirements efficiently

Forces
It is difficult to identify similar requirements that can be clustered for a generalized solution. A generalized solution has to define standards, while being flexible and considering possible variations.

Solution
To solve this problem, projects, areas or domains has to be identified, that have a similar or the very same scope (also see M-04: Domain Driven Design). To then define a template solution, the last three previous projects of an area or domain and their implementation are analyzed. Comparing the different steps and procedures, a standard that could have been applied to all those projects is developed.
After developing this standard, it is compared with a project where this standard wouldn't work. Going through the different steps, the question is asked, if this project is so important, that it would make sense, to adjust the standard closer to this particular project. After calculating all standards, a

template, that can be applied on all projects is implemented, written down and given to all relevant employees, working on the projects. There might be some projects, that have never been executed before: If this happens goal, the input and the outcome is analyzed. After that a process, that is as value-stream-optimized as possible, is developed. It might happen, that the standard can't be applied to a project, because it is to different. If this happens, the template and the project are compared with each other to analyze which interfaces can be standardized.

Variants
Instead of writing the template down and handing them to the employees, how-to videos are provided and uploaded on the intranet. The material is not provided in any written or recorded form, instead there are "experts" within the teams, consulting, if any questions or problems occur.

Consequences
Benefits:

- Saving time, because less work capacity is needed
- Saving money, because less consulting companies are used
- Applying the same standardized solutions, leads to less problems

Liabilities:

- Working on these standards need a high workload
- Standardization sets limits to the project, because it might not be the optimal solution

See Also
M-04: Domain Driven Design

C.2 Documentation of Methodology Pattern Candidates

C.2.1 Velocity Measurement

Pattern overview	
ID	M-01
Name	Velocity Measurement
Alias	–
Summary	Velocity measurement takes in information from previous sprints to calculate a forcast for how much work will be done in the next one. This can be done for single teams or overall.

Example
At DevelopmentCo the Product Owner managed an agile program. At the beginning of the program a deadline had been defined which had to be fulfilled. This collides with agile principles.

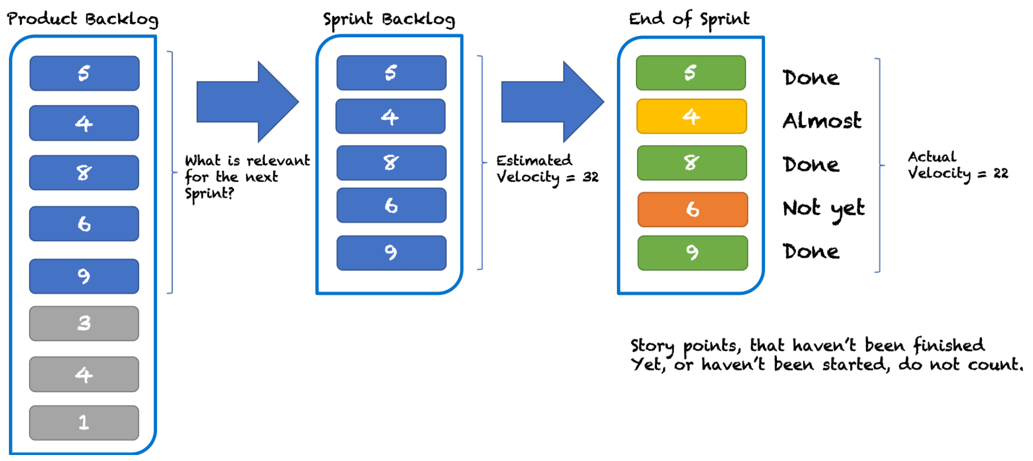
Context
This problem occurs whenever you manage your program agile.

Problem
The following concern is addressed by this M-Pattern:
C-103: Predictability in agile Program Management

Forces
The backlog is not completely defined in the beginning of the program but will change over time.
User Stories can vary in their complexity and thereby it is difficult to predict how many User Stories are being completed per sprint.

Solution
A prerequisite is that all items in the Backlog has to be estimated in Story Points.
A Sprint has to be performed normally based on the Sprint Backlog. At the end of a sprint the completed User stories of every team are being aggregated to calculate the total amount of story points. User stories that have been started but not completed are not being counted and will be carried into the next sprint. Once the total amount of story points for the last sprint have

been calculated the same thing is done for the last three sprints. With these figures the average amount of story points per Sprint can be calculated. This will help to predict the story point throughput per sprint.



Variants

If the different sprints by each team aren't synchronized you still calculate the story point throughput of each team and aggregate when the last sprint is done.

Consequences

Benefits:

- Planning security in how much work can be done per sprint
- Planning security in how long the program will last
- Comparability between the team to be able to identify very strong or very weak teams
- Increasing precision over time

Liabilities:

- A high amount of user stories that are just almost done can affect the accuracy

See Also

V-01: Burndown Chart

C.2.2 Continuously changing the improvement method

Pattern overview	
ID	M-02
Name	Continuously changing the improvement method
Alias	–
Summary	Going through the very same improvement process for too often can lead to a stagnation of the improvement in the team. Changing the improvement method can lead to new perspectives, more creativity and thereby more improvement.

Example
At DevelopmentCo, a development team was in a long-term project. During their Retrospectives, some topics were coming up every time. Either because they were very ambitious or their value add wouldn't justify the effort, the suggest topics never came to fruition. But talking about the same topics over and over again was stagnating the improvement process in the team.

Context
This problem occurs whenever members of the development team go through the same improvement process for too often. For instance, if the same problem or topic comes up too often in a retrospective, people won't react on it or improve it.

Problem
The following concern is addressed by this M-Pattern:
C-109: Continuous improvement remains static after a certain amount of time

Forces
To improve something means to change something. But it's not easy to constantly change something while being stuck in the same process.

Solution
To get a better mindset for the people to actually change something in their work habits, you also have to change the improvement process. Here are various options to switch up things. Frequently new improvement processes should be applied to help think out of the box. Here is a list of possible improvement processes:

- Rapid Improvement Events
- Value Stream Mapping
- 5 Why's
- DMAIC

Variants
Employee transfers between teams can also lead to an improvement. New people bring a new mindset to the team.

Consequences
Benefits:

- The continuous improvement process can improve itself
- Trying out different versions of an improvement process will also show what works best for what team

Liabilities:

- People don't like change. If one improvement process works for one team and then it's changing again, can also be destructive

See Also
-

C.2.3 Mapping storypoints to other KPI's

Pattern overview	
ID	M-03
Name	Mapping storypoints to other KPI's
Alias	–
Summary	Storypoints are fictive and only make sense within one development team. By mapping them to other KPI's, they have more meaning and can be used for continuously calibrating the forecast instead of referencing user stories.

Example
At ConsultingCo, several agile projects have been implemented already but they were not able to use this knowledge for estimating upcoming projects in time and budget. The problem was that they were estimating their User Stories in Story Points, which is a fictive number that shall not have any further meaning.

Context
This problem occurs whenever User Stories are being estimated with Story Points and shall further on be used to calculate time, effort or any other key figure based on the Story Points.

Problem
The following concern is addressed by this M-Pattern:
C-108: Story Points are not comparable outside teams/projects

Forces
Story Points are meant to be fictive and are not related to any rational key figure like time.
Several teams can have a different understanding on what amount of Story Points matches what time and effort.

Solution
Story Points can be used for estimation if the associated story is being referenced to another key performance indicator (KPI). For instance, if the Story Points shall make a statement about the time of a future story/feature/project, the referenced KPI must be about time as well. Possible KPI's can be the time used for a user story or the Story Points estimated within on sprint, where a sprint is a defined period of time.

These references will be used for further predictions.

Variants
Any KPI's that can be related to Storypoints can be used as reference for further predictions.

Consequences
Benefits:

- Making more use of a key figure already being collected.
- Helping predictions to be more precise.

Liabilities:

- Story Points are meant to be fictive and any developer should not map them to any other figure like costs or time. But this mapping in the head can happen very fast and is hard to undo

See Also
-

C.2.4 Change Backlog

Pattern overview	
ID	M-06
Name	Change Backlog
Alias	
Summary	The Change Backlog is a set of items that describe tasks regarding an organizational change. As any other Backlog it is the basis for a Sprint Backlog, which in this case assures that organizational changes are implemented step by step.

Example
A large sized company was planning an organizational transformation to break down internal silos. Therefore, they estimated a three-year duration and a change roll-out each year. So, one roll-out was prepared and planned for 12 months until all changes were applied at once. The results weren't as good as expected, mainly because of a strong resistance by the employees.

Context
This problem can occur whenever large organizational changes are applied.

Problem
The following concerns are addressed by this M-Pattern:
C-19: Dealing with internal silos
C-117: Emotional impact of Agile Transformations to employees

Forces
To break down internal silos, the structure of the organization has to be changed, which is a difficult procedure.
The bigger the change, the higher the risk of employees offering resistance.

Solution
An organizational change or a transformation will be performed by a team of experts. E.g. when trying to break down internal silos, one expert per silo will join the team, making it a cross-functional team. When planning an organizational change, it should start with identifying tasks that have to be performed. These tasks are managed in a Backlog, with everything that a Backlog requires, like estimating tasks or prioritize them.
Afterwards this Backlog will be the basis to a Sprint Planning. Sprints regarding an organizational change will differ in their timeframe though, being

longer than a Sprint in e.g. software development. The entire change or transformation will thereby be performed using agile practices.

Variants
-

Consequences
Benefits:

- Sustainable change
- Smaller changes will increase the success as well as the acceptance
- Problems or mistakes can be identified and fixed faster

Liabilities:

- Applying all agile principles might not be reasonable at this point. Planning a change doesn't need to be completely transparent

See Also
-

C.2.5 Value stream analysis

Pattern overview	
ID	M-07
Name	Value stream analysis
Alias	–
Summary	A value stream is a assembler for value just like its organization and delivering. Using this might help to break through the silos.

Example
A consulting company has different lines like design, data, event, consulting and training. The problem is, that the different lines don't work together properly, which would make a huge difference, because not cooperating with the other lines leads to only bailing out the potential within one silo and not all of them.

Context
This problem appears whenever functional units within a company's hierarchical structure are only focused on their own function but lacking interfaces to other units.

Problem
The following concern is addressed by this M-Pattern:
C-19: Dealing with internal silos

Forces
Cooperating between the different lines or silos might be difficult because there is sometimes still a rivalry between them, which leads to not wanting the others to gain any profit. To break down the barriers between the silos, it is required to obtain the management buy-in on a high strategical level. Depending on one's position within the company's hierarchy, this can be very difficult to achieve.

Solution
A value stream is an assembler for value just like its organization and delivering. Using this might help to break through the silos.
To start with, a representative of each silo meets up to discuss the different value streams. First it is necessary to analyze the value stream of each silo by its own, and after that identifying the interfaces between all silos, to see how and where the different silos are connected. By doing that, it should be

possible to connect each and every silo together and see all connections in the Value Stream. This helps to not only see the work of one silo but to draw the whole picture and gain a higher consciousness between the silos and the employees.

Variants
Instead of representatives of each silo, the value stream analysis can also be conducted by external consultants. This might help to when dealing with internal office politics.

Consequences
Benefits:

- See the whole picture and not only the silos
- Increase the motivation by showing the employees how working together can actually increase the outcome

Liabilities:

- - The analysis takes some time and because there is only one representative of each silo, it could be difficult to communicate the outcome of the analysis

See Also
-

C.2.6 Nexus Sprint

Pattern overview	
ID	M-08
Name	Nexus Sprint
Alias	–
Summary	The Nexus Sprint is an Artifact of the Nexus Framework and helps to scale the original Scrum Process to large-scale agile development. One big advantage is, that the several development teams do not necessarily have to synchronize there Sprints.

Example
At a company, they ran a program and divided the teams by the applications that had to be maintained. These applications had fix release cycles, but every application had a different release cycle, which made it difficult for the teams to synchronize their sprints.

Context
This problem occurs whenever outer circumstances have an influence on some sprint periods.

Problem
The following concern is addressed by this M-Pattern:
C-78: Synchronizing sprints in the large-scale agile development program

Forces
Some outer circumstances like dedicated release cycles for applications are fix and cannot be changed. The result of a sprint is a release-ready increment, but if it doesn't fit the release cycle of the related application, it is not really deployable.

Solution
Development teams can apply a nexus sprint. This form of the sprint planning does not necessarily require the teams to synchronize their individual sprints, even though it is suggested.
If several Sprints have to be performed asynchronous, the Product Owner has to defined Sync Points. These define a strict timeframe after which a release-ready increment is being created. That means, that Sync Points define an overall Sprint for the entire program. Team Sprints don't have to synchronize with the Sync Points, but the Product Owner has to make sure,

which Backlog Items will have to be ready until the next Sync Point.

Variants
-

Consequences
Benefits:

- Teams are able to work more independent
- More flexibility in planning and structuring the program

Liabilities:

- Scales only up to 9 teams
- More coordination effort to the Product Owner

See Also
-

Other Standards
The nexus framework is a scaled form of the Scrum, developed by the same authors.

C.2.7 Portfolio Backlog

Pattern overview	
ID	M-09
Name	Portfolio Backlog
Alias	Strategical Backlog
Summary	The Portfolio Backlog is a Backlog on the highest hierarchal level of an organization, managing strategical initiatives using agile practices.

Example
At a large company, the IT organization was working by agile values and principles using the SAFe framework. The company's strategy on the other hand was located in a traditional organized department and delivered results once a year. That restrained the IT Organizations strategic level to fully follow agile principles.

Context
This problem occurs whenever the operational level follows agile processes while the processes on the strategical level are not aligned.

Problem
The following concern is addressed by this X-Pattern:
C-113: Disconnect between strategy and operation

Forces
The need for becoming agile usually comes from the operational level, which is why the transformation usually starts there as well. But since the strategical level is located higher within the hierarchy, the operations has no authority to make them align their processes. Applying an agile framework for large companies reaches far beyond team or program level, but fully aligning all processes takes time.

Solution
Agile practices can be adopted through the entire organization up to the highest hierarchical level, which helps to align all agile processes. The Portfolio Backlog is a set of business and enabler Epics that have been approved and prioritized by the Portfolio Management. It receives direct impact from the organizations vision and will help the organization, their partner and customers to plan with upcoming releases. So the Portfolio Backlog can be seen as a strategic roadmap. It's epics will await implementation and will

later move forward to an Agile Release Train or Solution Train. By handling high-level strategic topics using agile practices, it perfectly aligns with the rest of the organizations agile processes. Also will the estimation of large strategical topics be easier than usual, because defining them as Epics makes it possible to estimate them based on Storypoints. This will be based on the collected data from previous projects, making the estimation more precise.

Variants
Such large Epics are sometimes also defined as a Saga, to illustrate the complexity of the initiative. A Saga would then again contain several Epics. But this approach is also controversial. All items within a Backlog are in a way User Stories, having the very same characteristics. Defining a large hierarchy for Backlog items will not necessarily add value but could also confuse instead.

Consequences
Benefits:

- - Alignment to other agile processes
- - Better forecast for strategical initiatives
- - More transparency of the organization's strategy

Liabilities:

- Agile practices must be fully understood, otherwise this pattern will lose its benefits

Other Standards
The Portfolio Backlog was introduced by the scaled agile framework (SAFe)

C.2.8 Weighted shortest job first

Pattern overview	
ID	M-10
Name	Weighted shortest job first
Alias	WSJF
Summary	Weighted shortest job first is a technique to prioritize the Backlog by calculating the value of each item. It thereby follows the theory, that item of the same value with lower estimated time are more valuable because their value add can be faster done.

Example
During the implementation of a project, ConsultingCo was facing the problem that the higher management of their customer disagreed with the structure of the project, meaning the prioritization of some items and why they would have to wait so long for some results.

Context
This problem occurs whenever there is a communicational gap with stakeholders or when processes between strategy and operation are not enough aligned.

Problem
The following concerns are addressed by this M-Pattern:
C-44: Dealing with communication gaps with stakeholders
C-113: Disconnect between strategy and operation

Forces
Agile practices are mainly used on the operational level and not the management level. That is why the management might not understand how agile practices are being applied.
There are a lot of communication problems, because most of the time there is only one person functioning as an interface between these two levels, which can lead to a loss a information or misinterpretation.

Solution
A precondition is, that the tasks have been estimated already. By calculating the cost of delay and the duration of each job, it is possible to find out, which task should be done first.

Cost of Delay = User-Business Value + Time Criticality + Risk Reduction and/or Opportunity Enablement
To calculate the duration you can use different patterns, just like Magic Estimation or T-Shirt Sizes. After calculating these two values, the cost of delay is divided through the duration of a task.
The outcome of this shows, which task is the weighted shortest job, that should be done first.

Variants
There are different patterns that can be used to calculate the duration of a job, so as Magic Estimation or T-Shirt-Sizes or any other method to measure the duration of a task.

Consequences
Benefits:

- Triggers an active information exchange between strategy and operation
- Prioritizing jobs based on the economic efficiency
- Easy and quick to prioritize new backlog items and classify them, because there is no need to discuss it

Liabilities:

- The calculation is based on a subjective measurement, which means, there is still some uncertainty left

See Also
-

Other Standards
SAFe 4.6

C.2.9 Improvement Backlog

Pattern overview	
ID	M-11
Name	Improvement Backlog
Alias	–
Summary	Code quality can vary because of many reasons. If experienced developers see the possibility for code improvement while working on a different task, they don't always manage to implement it right away without threaten the completion of the sprints goal. Therefor the should have a distinct place for documenting such improvements for later or for others to implement.

Example
At DevelopmentCO different developers have more or less experience. It happened various times that experienced developers were working on one of their tasks while identifying a possible improvement in someone else's code. Sometimes even in the code of another teams' team member. Unfortunately, they were not able to implement the improvement right away without jeopardiesing the goal of the current sprint.

Context
Whenever improvements of medium or high effort are being detected.

Problem
The following concern is addressed by this M-Pattern:
C-114: Maintaining equal quality among teams

Forces
Possible improvements that are neither implemented right away nor documented can easily be forgotten. The effort for implementing an improved solution can vary and thereby sometimes exceed the developer's capacity. A developer always has to prioritize the tasks in the current sprint and thereby the current sprint goal over improved solutions.

Solution
Whenever a developer identifies a possible improvement in the code, he will follow these steps:
"Do I have free capacity to implement the improvement right away?"
o Yes: Implement the improvement

o No: Create a task in a separate Improvement Backlog and document the steps for the improvement

Thereby it doesn't matter if the improvement was identified in one's own code, or someone else. An identified possible improvement could be an unclean code, performance issues or others.
During the Retrospective new items in the Improvement Backlog are being presented and discussed in the team.
The Improvement Backlog can provide tasks during a sprint planning or can be considered by developers with free capacity if the Sprint Goal is not in jeopardy.

Variants
-

Consequences
Benefits:

- Encourages an active knowledge exchange among teams and team members
- Helps to assure a constant quality among all teams
- Separates improvement tasks from other tasks to not blow up the Backlog

Liabilities:

- Could lead to rather document tasks than doing them right away
- Could be, that the improvement Backlog is growing way faster than its tasks are being solved as well

See Also
-

C.2.10 Mob-Testing

Pattern overview	
ID	M-12
Name	Mob-Testing
Alias	–
Summary	Mob-Testing is a way of manual testing the UI of an application. It gives great insights about how the user actually interacts with the UI. Mob-Tests can also be used as a demo for stakeholders to watch or participate, which involves stakeholders more in the project and also leads to a better feedback for the development team.

Example
At ConsultingCo a business department requested the implementation of a new software tool. This program was implemented using agile methods. After implementing the first MVP the stakeholder was introduced to the software and his feedback was mainly positive. After a second milestone the stakeholder was not satisfied at all and was criticizing aspects, he could have already seen in the first MVP.

Context
This problem occurs whenever key stakeholders are not enough involved in the progress of a program.

Problem
The following concern is addressed by this M-Pattern:
C-44: Dealing with communication gaps with stakeholders

Forces
Stakeholders expect that their needs are being implemented according to their wishes and therefor don't see a need to constantly follow the process. Most stakeholders want to spend the least time possible on a program with the biggest valued possible.

Solution
A Mob-Testing is an event where all developers or some representatives of every development team including Scrum Master, Product Owner and Product Manager come together to extensively test the UI on selected features. During this event there are four different roles, that the different participants can step into. The four roles are: the driver, the facilitator, the navigator

and the mob. The driver will receive a task from the navigator which he has to solve. A task constitutes an end-to-end functionality. An example would be: “create a new user in the system”. The navigator is not allowed to give any hints on how to solve the task or how to navigate through to the correct menu. Thereby the UI will be tested on its usability. During this whole time the facilitator checks if all rules are complied strictly. If the driver has a problem or can’t find what he is looking for, he is allowed to ask the mob some questions. The mob consists of all the other participants. After every tested functionality the role of the driver and the navigator changes. The facilitators stay the same during the event. The main role of the mob is to observe the behavior of the driver to gain feedback on how the software will be used.

Variants
A Mob-Demo follows the very same structure as the Mob-Testing event with the only difference, that the key stakeholders are invited as well, who will find themselves in the role of the driver or a member of the mob. Thereby the key stakeholders get detailed insights on the implemented software and developers are able to observe hands-on experience of the potential customer which will give them valuable feedback.

Consequences
Benefits:

- Involving key stakeholders
- Valuable feedback for developers
- Knowledge exchange between developers and stakeholders
- Stakeholders might find defects, that the developers haven’t detected yet

Liabilities:

- Time-consuming
- Finding to many defects might cause a lack of confidence in the project or the developers

See Also
P-06: Proactively involve key stakeholder in the progress with every increment

C.2.11 T-shirt size estimation

Pattern overview	
ID	M-14
Name	T-shirt size estimation
Alias	–
Summary	Estimating a new demand can be time consuming which is unnecessary in many cases. In a very early stage – usually before the project starts – a very rough estimation delivers all informaiton needed. In this case t-shirt sizes (S, M, L, XL) are being mapped to a period of time for how long the project may takes.

Example

The department for demand management at InsuranceAG noticed, that they spend a lot of time estimating new demands. A request for demand is raised whenever another department wants to start an IT supported project. But based on the estimated time and costs, many requests were withdrawn, which makes the work on a very detailed estimation useless.

Context

Whenever there is a request for demand, it will not necessarily evolve into the start of a project.

Problem

The following concern is addressed by this M-Pattern:
C-116: Estimation of complex demands/requests

Forces

Sometimes a request for demand is placed to gain more information about the complexity of the project without the intention of starting it.
Performing full requirements engineering is the perfect basis for a very concrete estimation, but too much effort when it's not clear whether the projects actually comes to fruition.

Solution

There are normally five different T-Shirt Sizes: XS, S, M, L, XL. XS is the smallest and XL the biggest. Not those sizes are assigned to every initiative to estimate how small or big it will be.

Small tasks, that won't cost much or only need a short amount of time, get size XS or S, the bigger the task, the bigger the size. There is no mathematical formula to define what defines the different sizes, but there is a least a general comprehension. After defining the different sizes for the project, it is necessary to map them to a distinct time frame and budget.

Variants
Sizes can be given different values, depending on how big the projects are, that the company has.

Consequences
Benefits:

- Saving of time Premature estimation helping to assess the extent of a certain project

Liabilities:

- Less precise
- Time-consuming

See Also
-

C.2.12 Agile Ninja

Pattern overview	
ID	M-15
Name	Agile Ninja
Alias	–
Summary	The Agile Ninja is a variation of an Agile Coach. This role is applied whenever there are organizational restrictions because of which the Agile Coach cannot fulfill his role completely.

Example
At LogisticsCo they implemented a program, where they defined an agile coach for every team. Because of capacity problems the agile coach had to take on technical tasks. That was leading to that the agile coach not being independent anymore, what entailed, that he couldn't live out his role properly. This caused problems, because his guidance wasn't impartial anymore.

Context
The problem occurs whenever roles wrongly defined or misunderstood.

Problem
The following concern is addressed by this M-Pattern:
C-56: Defining clear roles and responsibilities

Forces
For many agile roles there is superficial knowledge which is why the pure name can lead to misunderstandings.
Not every organization gives the opportunity to act out a role exactly as defined in a framework.

Solution
Due to outer circumstances given by the organization structure it is not always possible to implement the role of an Agile Coach within an agile team. If such restrictions are given, a similar role needs to be defined, making sure no misunderstandings exist, based on the name of the role.

The Agile Ninja is a part of an Agile Team and takes on tasks like any other team member, but he is also especially skilled in agile practices and functions as a contact person regarding questions on the agile methods.

The Agile Ninja can be compared with an Agile Coach, but the difference is, that he cannot be completely impartial, due to the fact that he has responsibilities just like any other in the agile team.

Variants
–

Consequences
Benefits:

- Not only a team member but also a coach
- One distinct contact person for Agile Methods

Liabilities:

- Not completely impartial

See Also
–

C.2.13 System Thinking

Pattern overview	
ID	M-16
Name	System Thinking
Alias	
Summary	System Thinking is a theory that aims to describe the basic functioning of complex systems. It thereby goes through the two phases of Analysis and Synthesis to first focus on each component within the system and furthermore their relationships within the system.

Example
A consulting company has different lines like design, data, event, consulting and training. The problem is, that the different lines don't work together properly, which would make a huge difference, because not cooperating with the other lines leads to only bailing out the potential within one silo and not all of them.

Context
This problem appears whenever functional units within a company's hierarchical structure are only focused on their own function but lacking interfaces to other units.

Problem
The following concern is addressed by this M-Pattern:
C-19: Dealing with internal silos

Forces
Cooperating between the different lines or silos might be difficult because there is sometimes still a rivalry between them, which leads to not wanting the others to gain any profit. To break down the barriers between the silos, it is required to obtain the management buy-in on a high strategical level. Depending on one's position within the company's hierarchy, this can be very difficult to achieve.

Solution
The solution for breaking through those silos is system thinking. Here the focus is not just on one line, it tries to see the whole system and analyze the relationships between the different lines, rather than just focus on one. There are two approaches to describe the system. The first one is to analyze

and describe the individual components and find out what they can do. The second one is the synthesis which describes the relationship between those components so that it is possible to not only view the possibilities of each individual but to figure out how they can work together.
That does not only help to gain new insights by getting the view of all different lines on one project, but also increases the communication between them and the outcome for the whole company.

Variants
Depending on how big the company is, it might be different how deep you decompose the silo into components.

Consequences
Benefits:

- Gaining a higher outcome for the company
- Increasing the communication within the different silos
- Interdependences become transparent

Liabilities:

- Analyzing the components and their relationships is a time-consuming process
- Implementing the identified changes to the organization will be a high level change and thereby critical

See Also
-

C.3 Documentation of Viewpoint Pattern Candidates

C.3.1 Burndown Chart

Pattern overview	
ID	V-01
Name	Burndown Chart
Alias	–
Summary	The burn chart visualizes the velocity of a program and thereby shows weather the program is within the estimated time schedule.

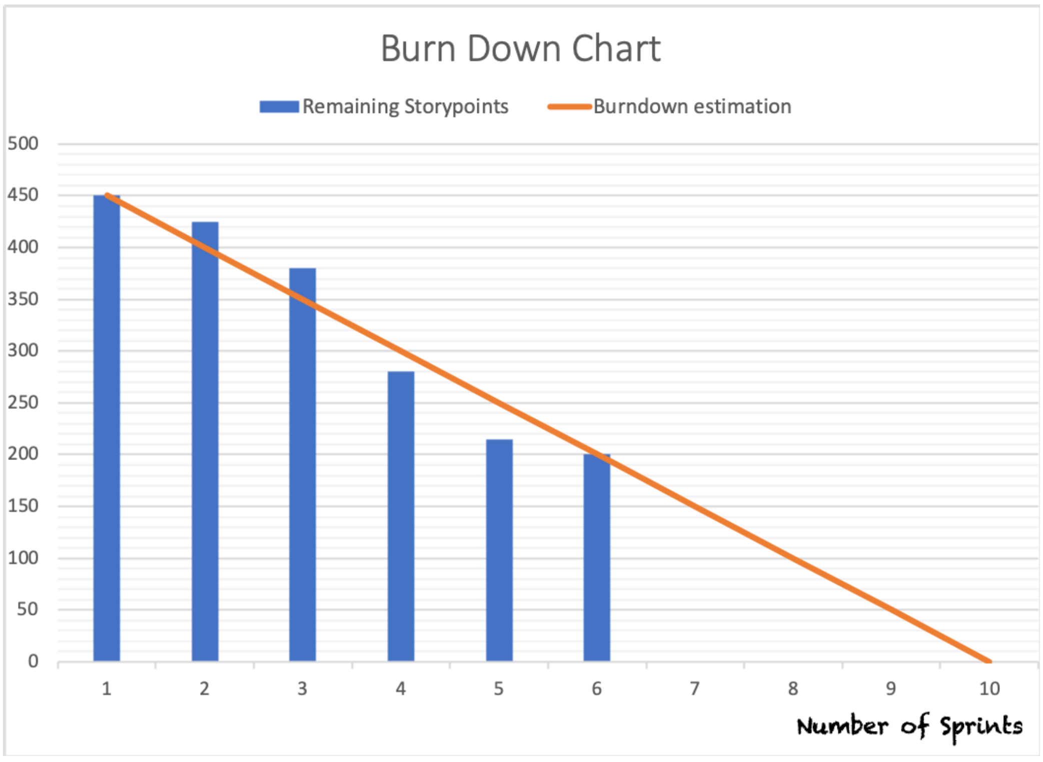
Example
At DevelopmentCo the Product Owner managed an agile program. At the beginning of the program a deadline had been defined which had to be fulfilled. This collides with agile principles.

Context
This problem occurs whenever you manage your program agile.

Problem
The following concern is addressed by this V-Pattern:
C-103: Predictability in agile Program Management

Forces
The backlog is not completely defined in the beginning of the program but will change over time. User Stories can vary in their complexity and thereby it is difficult to predict how many User Stories are being completed per sprint.

Solution
The Burndown Chart tries to predict the overall time needed to complete the program. The x axis describes the time in sprints while the y axis describes the amount of remaining Story Points. At the end of each sprint, a new point of remaining story points is being added to the chart. The red dashed line predicts the end of the program (see figure).



Variants

-

Consequences

Benefits:

- Visualization of the projects time course

Liabilities:

- The explanatory power decreases if there have been changes to the capacity of one or more teams

See Also

M-01 Velocity Measurement

Other Standards

-

Data Collection
The amount of remaining Story Points is being documented in the Backlog. The number of Sprints is based on the program's deadline (if given). The single points for the Story Point history within the charge is being calculated by the Velocity Measurement.

C.3.2 Storymap

Pattern overview	
ID	V-02
Name	Storymap
Alias	Program Board
Summary	A storymap can help visualize the overall project scope as well as dependencies among user stories and can easily be adjusted if the scope changes.

Example
At ConsultingCo they pried open all Epics into different User Stories and every team got an area of responsibilities. After merging the solutions of several teams, the outcome wasn't always as expected, because the individual teams didn't have the context that they would have needed.

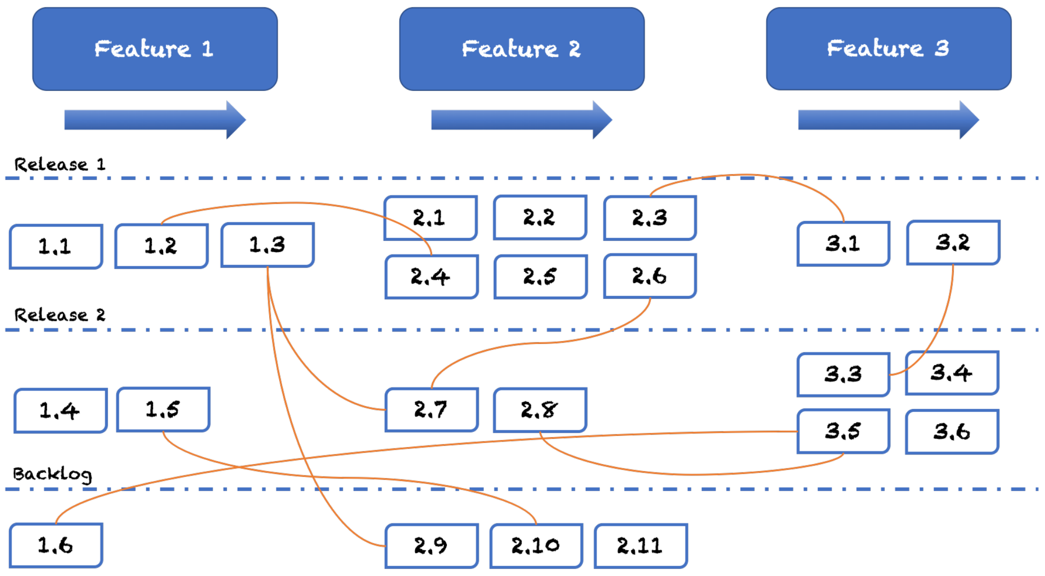
Context
The more teams work on the same project, the more likely this problem occurs.

Problem
The following concerns are addressed by this V-Pattern:
C-118: Establish a common vision of the product
C-126: Size and complexity of the backlog

Forces
It is difficult to fully communicate the product vision to every member of each team, but in many situations, it is important for the developer to understand the full context of a project. Large Scale Projects usually contain a lot of Epics, User Stories and Tasks, but the more items the backlog contains, the more difficult it is to manage and maintain it.

Solution
With a growing number of items, the backlog gets too complex. To prevent this problem, it is necessary to visualize all user stories, to not only get a better overview, but also show the dependencies between the different items to maintain the customer focus. This is called a User Story Map and does not only show the order but also which features the different tasks belong to and how important the User Story is.
The first step is to write every User Story on one card or piece of paper and pin it onto the wall. Be doing that a "map" with an overview of the whole

project is created. After that all cards are sorted after priority (vertical) and the time when it has to be finished (horizontal). The last step is to point out the dependencies by connecting the relevant User Stories with each other.



Variants

The considered level can vary, based on how specified the Backlog is and on the current focus. So instead of User Story level, it could be more roughly on Epic level or more detailed on Task level. It is not only possible to make a physical board, but also to create it online, so everybody has an easy access.

Consequences

Benefits:

- Helping everyone to better understand the backlog and thereby the projects scope
- Good oversight because of the visualization
- Allows to break down the different items into smaller ones
- Easy to view all dependencies
- Encourages the communication of the team members

Liabilities:

- Time-consuming
- Steady updating required, otherwise the board loses its value
- If it is a physical board, its only available for the team members, that are close to it. Provided photos of the board, would have to be updated all the time, to make sure everyone is up to date.

See Also
CO-01: Velocity Measurement

Other Standards
-

Data Collection
The Items for the Storymap come from the Backlog. At one planning event, attended by Development Teams, Scrum Masters, Agile Coaches, Product Owners, Product Managers and/or Stakeholders, the items will be ordered chronologically and dependencies will be identified.

C.4 Documentation of Anti-Pattern Candidates

C.4.1 Don't let the Product Owner be the only interface to the team

Pattern overview	
ID	A-01
Name	Don't let the Product Owner be the only interface to the team
Alias	–
Summary	If the outer circumstances of an agile project are changing, then the project has to adapt to that very change which can be a minor detail or fundamental different scope. The product owner is the only person who can bring this change into the project since he is the owner of the projects Backlog.

Example
At ConsultingCo, due to the structure of the company and the current project, the Product Owner was the only instance that received information from stakeholders, passing them on to the team. At one point the overall scope for the project changed. But due to misunderstandings or disagreement, the Product Owner did not apply the changed scope correctly to the Backlog, which lead the project to the wrong direction.

Context
This problem can occur when the projects scope changes.

Problem
The following concern is addressed by this Anti-Pattern:
C-101: Product Owner is the only instance to the team

Forces
If the scope of the project is changing, you never know how the Product Owner is reacting to it, communicating it to the team and implementing it into the backlog.

General Form
At the beginning of every project the overall scope is being communicated

and all stakeholders align themselves to commit to the projects goal and how to get there. During the projects implementation outer circumstances can change, forcing the project to do a fundamental change to its scope. A new scope is communicated top-down and the actual change in the Backlog is only being implemented by the Product Owner. If he/she does not approve with the new scope or misinterprets is the teams will still follow their backlog and thereby the Product Owners perspective.

Consequences
If one or several development teams are not reacting to the new project scope, the projects business case can partly or fully get lost.

Revised Solution
Have in general a second communication channel next to the Product Owner, for instance the Agile Coach. That way you not necessarily take away authority or power from the Product Owner but have another control instance to assure that the project is always going the right way.

See Also
-

Other Standards
-

C.4.2 Don't think a change in too big steps

Pattern overview	
ID	A-02
Name	Don't think a change in too big steps
Alias	–
Summary	Big changes in an organization can cause negative effects. The bigger the change, the more likely it is, that the employees, who will be affected most by the change, will offer resistance. It is better, to implement changes in small steps, to make them sustainable.

Example
After identifying silos in a company and trying to break them open, they defined a goal that they wanted to achieve. They decided to adapt the change immediately with only one small interims solution, meaning that they applied a huge change in a short time.

Context
The problem appears, whenever the management of a company tries to counteract against internal silos with adapting change too quick and it is not accepted by the employees.

Problem
The following concern is addressed by this Anti-Pattern:
C-19: Dealing with internal silos

Forces
When planning a change, it sometimes appears difficult to break it down into smaller steps. The bigger the applied change, the higher the chance for employees offering resistance to it.

General Plan
Implementing a change in the daily operations of a company can be challenging. Changing a running system isn't something that can be done overnight. Transform a system and develop a new way of doing things need time and a lot of effort. Choosing a new way of action and implement it without making adjustments or listen to what the employees want will not make all problems go away, it will rather cause even more problems. Change a system and make it work needs a lot of effort and interim steps, so that the employees can get used to the new way of doing things and to make adjustments if necessary.

It can even take several years to change a system properly and make it work.

Consequences
Benefits:

- Fast change/transformation

Liabilities:

- Risks the success of a change/transformation
- Increasing risk for resistance offered by the employees
- Current situation of the organization can get worse instead of better

Revised Solution
By downing the change into smaller steps and adapting the changes in smaller cycles.

See Also
M-06: Change Backlog

C.4.3 Don't instate a field specialist as Product Owner with no technical background

Pattern overview	
ID	A-03
Name	Don't instate a field specialist as Product Owner with no technical background
Alias	–
Summary	Sometimes when a business department is the driver for a new IT project, they want to gain more control to it by instating the Product Owner with an employee of their own department. But this decision is often not rational and will risk the project's success.

Example
At AutomotiveCo they instated someone from a specific department as Product Owner, thinking he would have a lot of useful insights. The problem was that he did not have enough technical knowledge which caused some bad decisions for the project.

Context
This problem occurs whenever the instated Product Owner has no technical background.

Problem
The following concern is addressed by this Anti-Pattern:
C-120: Product Owner lacking technical understanding

Forces
As the interface between the development teams and the customer, the Product Owner needs to have both professional and technical understanding.

General Form
In big companies the responsible department normally triggers a new IT project and thereby takes the role of the customer. As a financial sponsor the department tries to take a huge influence on the outcome of the project. One way to do so is to instate the Product Owner out of their own department. This influence is way to strong and this decision should not be made by the department, but by the IT instead. This often leads to a lack of technical understanding of the Product Owner which might be required in

key decisions throughout the project.

Consequences
Benefits:

- The Product Owner has a lot of professional understandig and can thereby define the tasks specifically.

Liabilities:

- Wrong decisions in key situations can have strong negative impact on the projects success.

Revised Solution
Staffing decisions should entirely be made by the IT department. The role of the Product Owner requires a basic technical understanding of the current project.

See Also
-

Other Standards
-

C.4.4 Don’t let teams work in the same constellation for too long

Pattern overview	
ID	A-04
Name	Don’t let teams work in the same constellation for too long
Alias	–
Summary	Continuous improvement is an important aspect of agile principles and there are many different improvement processes and methods that help follow this principle. But if a team constantly works in the very same constellation, the improvement of each individual team member will stagnate.

Example
At DevelopmentCo, a development team was consisting of the very same members for almost two complete programs, each with a duration of one year. Late during the second program a new developer from another team joined this one. Until then the development team wasn’t aware of having a problem in their continuous improvement. It was only realized when the new developer brought in new ideas, new skills and new creativity.

Context
This problem occurs when several development teams within a program are working very separate from each other without any structured exchange of knowledge.

Problem
The following concern is addressed by this Anti-Pattern:
C-121: Knowledge exchange between teams

Forces
People don’t like change, so developers will not necessarily want to switch teams. The problem is difficult to identify, because individual improvement is difficult to measure.

General Form
A development team that works in the very same constellation for 9 months or more will be a strong team and know each other very well. That might

include personal code style, undocumented naming specifics or any other common collaborations that should've been documented as a standard practice but there was no need identified within the team. That way they will perfectly learn how to collaborate with each other but not necessarily with other developers from other team, other programs or other companies.

Consequences
Benefits:

- The team will most likely be more effective than a completely new team
- Less effort for documentation of specific functionalities because it's just known within the team

Liabilities:

- The team will be tempted to document less
- The continuous improvement process can also be destructive if bad practices sneak in within a team

Revised Solution
Encourage developers to switch teams after a certain amount of time. Depending on different factors, stagnation can begin after 6 months. Never have the very same constellation of a team after one complete program or after more than one year. New developers will bring new thoughts to the team which will benefit the teams moral.

See Also
M-02: Continuously changing the improvement method in retros

Other Standards
-

C.4.5 Don't manage an unnecessary amount of requirements in one program

Pattern overview	
ID	A-05
Name	Don't manage an unnecessary amount of requirements in one program
Alias	–
Summary	When a program is tailored within the IT Portfolio Level, there is various different ways how to do so. But since a program is considered as complex anyway it might happen that the scope of a program is too big. This will blow up the Backlog and lead to an unnecessary complexity.

Example
At AutomotiveCo, the IT Organization used Domain Driven Design (see M-04) to define responsibilities within the organization but they were not able to gain all benefits of this pattern. One domain was containing a number of subdomains and each subdomain was still very complex with a lot of responsibilities. On Backlog was maintained per domain, which in some cases reached a complexity, that was very difficult to handle.

Context
This problem occurs whenever a program or domain is not tailored enough, so that the Backlog might contain a huge number of items.

Problem
The following concern is addressed by this Anti-Pattern:
C-126: Growing size and complexity of the Backlog

Forces
Sometimes domains are tailored based on shared resources and budget. In that case it is difficult to plead against the defined standard. Some programs or domains are difficult to split any further because of their dependencies within.

General Form
There are numerous reasons for why a Backlog is reaching a size, which's complexity is not efficiently to handle any more. It might be that the pro-

gram or domain is tailored badly. It might be that the responsibilities require this size. Or it could also be that the Backlog was growing over time. Nevertheless, at a certain size (which depends on the industry, the organization and other factors) the Backlog will lead to new concerns, which could've been avoided.

Consequences
Benefits:

- All in one place, keeping the number of contact persons low

Liabilities:

- Can lead to inefficiency
- One Product Owner might not be able to orchestrate all tasks within the Backlog

Revised Solution
Tailor a Backlog to a manageable size and adjust it at a later point if necessary.

See Also
-

Other Standards
-

C.5 Documentation of Principle Candidates

C.5.1 The Agile Connector

Pattern overview	
ID	P-02
Name	The Agile Connector
Alias	–
Summary	The agile connector is part of the high-level management in an organization and functions as an ambassador for everything regarding agile methods or the agile transformation.

Example

The management of ConsultingCo decided to transform the entire company to an agile organization. This process took a lot of time and energy and had his biggest concerns in dealing with doubts in employees and get everybody on board. The problem was, that many employees didn't know how and where to address their concerns about the transformation.

Context

This problem occurs whenever the high-level management is not perfectly aligned throughout the agile transformation.

Problem

The following concerns are addressed by this P-Pattern:
C-95: Missing orientation of leadership
C-110: Spread agile mindset in entire organization

Forces

Agile methods lead to more self-organized working which also means a shift in the way of leadership. The old pyramid structure for hierarchy and the command-and-order leadership style does not align with the agile manifesto. So high-level managers will have to define their role differently.

Solution

A successful agile transformation has to be driven from the high-level management. So there needs to be a very good understanding of agile methods and what this transformation means for the organization. Therefor one or a few members of the high-level management should take the role of an Agile Connector. This role is responsible for communication and knowledge exchange throughout the whole company and further on to customers and suppliers as well. Important requirement for this role is knowledge and experience in agile methods, coaching and leadership.

Variants
-

Consequences
Benefits:

- Employees know exactly where to address their concerns
- One or a few people of the higher management will drive the transformation very much forward
- A transformation can appear very abstract. But with Agile Connectors it becomes more tangible

Liabilities:

- Depending on the number of requests, it can be time-consuming for that very manager

See Also
-

Other Standards
-

C.5.2 Culture of empowering decision making

Pattern overview	
ID	P-04
Name	Culture of empowering decision making
Alias	–
Summary	This principle focusses on establishing a culture which helps empowering employees to make decisions and handle situations self-organized teams can be exposed to.

Example
At ConsultingCo, a large-scale agile software development project was executed. The Product Owner was managing the Backlog and the development teams were implementing their items from the Sprint Backlog. After having implemented an item, it was the common understanding that this item wasn't adding value. The related development team had doubts about the item beforehand but didn't say anything.

Context
This problem might occur when managing self-organized teams.

Problem
The following concern is addressed by this P-Pattern:
C-74: Empowering agile teams to make decisions

Forces
Making decision is a skill that have to be learned, not every employee will have that skill right away. When employees got used to following orders by their boss, being a part of a self-organized team will put them into new situations, that they have to deal with.

Solution
When traditional teams have a project leader, that instructs the team, telling them what to do, the team members have to think less about the big picture and just follow instructions. Self-organized teams have no hierarchy and have to figure out these things by themselves. Teaching employees in self-organized teams how to handle new situations and how to be able to make decisions or to bring up a topic is best be done by establishing the right culture to it. It starts with the Product Owner, Scrum Masters and Agile Coaches. These are the ones establishing a culture of empowering decision making. This is done by clearly communicating:

- Don't ask. Just do it.
- Every input is valuable
- You are entitled to a different opinion
- As managing personnel, we can make mistakes to. Tell if you don't understand a decision.

These general rules have to have to be a common understanding within the entire project. Especially passive teams, who are more reserved in their communication need more focus in setting these guidelines.
Second step is the teaching and mentoring within a team. Agile teams should always be stuffed in a way, that people with little experience work together with people with a lot of experience. This means the experience in soft skills, making decisions, bring up new topics or addressing a problem. That will constantly help people to grow as fast and as much as possible.

Variants
-

Known uses
-

Consequences
Benefits:

- Enables an active exchange among the team
- Helps the team to grow together

Liabilities:

- With every team member, it takes time to teach such soft skills
- While learning such soft skills, team members might make unfavorable decisions

See Also
-

Other Standards
-

C.5.3 Intercultural team building

Pattern overview	
ID	P-05
Name	Intercultural team building
Alias	–
Summary	A group of people working on the same thing does not automatically become a team and the distance is an additional challenge for cross-shore agile teams. But this can be tackled by flying to each others locations, meeting each other in person and actively working on becoming a team.

Example
At InsuranceCo, an agile team from India was added to a running program but the collaboration was not as successful as expected. The communication and commitment on several tasks did not go well based on (what they assumed) was a different mindset.

Context
This problem occurs whenever agile teams from different countries and especially different cultures are collaborating together in an agile program.

Problem
The following concern is addressed by this P-Pattern:
C-61: Dealing with cultural differences between cross-shore agile teams

Forces
Different cultures have a different way of doing their work and they might have a different expectation. Geographical distance makes team building difficult.

Solution
To tackle this problem, several different actions or events can help to break down cultural barriers and help every individual in the group to identify as part of a team. To help grow the team together, it is useful to organize teambuilding events, like workshops, a dinner or even a weekend getaway with all team members to enable cultural exchange also outside the office. Workshops will help to gain a common vision of what to accomplish as a team and at the same time the team members will learn about their cultural

differences or the way how things in the other country is done, helps understanding the other teams better. Therefore, it is important that this process is bidirectional, meaning that not only inviting a team but also visiting a team is important, to see both sides. This can have a positive impact on how the colleagues work together and show understanding for their foreign team members.

Variants
If it is not possible to meet in person, due to limited time or a limited budget, weekly online video chats would be compensation instead.

Consequences
Benefits:

- Breaking through cultural barriers and learn to understand the different way of thinking and doing things.
- Connect with colleagues over the world

Liabilities:

- Time consuming
- Maybe not useful for a project, that's limited for a short period of time

See Also
-

Other Standards
-

C.5.4 Proactively involve key stakeholder in the progress with every increment

Pattern overview	
ID	P-06
Name	Proactively involve key stakeholder in the progress with every increment
Alias	–
Summary	In traditional methods like waterfall, stakeholders are used to define a set of requirements and then being rarely involved during the implementation process. With agile methods stakeholders are more frequently consulted and need to be involved more in the current status and progress of the project.

Example
At DevelopmentCo, an agile program for the implementation of a SaaS solution was conducted. Some key stakeholders were adding new requirements throughout the program. The problem was, that their requirements did not fit into the logical structure or the user interface of the application at that very point. They simply didn't see the bigger picture and the value add they were hoping for would've been way less if the requirements were implemented accordingly, without checking back.

Context
This problem occurs whenever key stakeholders are not following the progress of a project as much as they should.

Problem
The following concerns are addressed by this X-Pattern:
C-44: Dealing with communication gaps with stakeholders

Forces
Stakeholders that are not familiar with agile practices don't understand, that their active participation throughout the project is very much required. Most stakeholders want to spend the least time possible on the project with the biggest value add possible.

Solution
There are two prerequisites for involving stakeholders into the programs

progress:

- all information necessary are available to the stakeholders. Therefor also see P-01: Fully transparent agile project
- The current version of the project must be deployed regularly, for instance by a CI/CD pipeline

Stakeholders must regularly be exposed to the current version of the project. That can be done by:

- Mailing the current deployed version including a link to it
- Inviting stakeholders to the Retrospective
- Inviting stakeholders to an exclusive meeting
- Inviting stakeholder to participate in a mob-test (see M-12: Mob-Testing)

Variants
If stakeholders are invited to a Retrospective, this can also be realized by splitting this meeting into two parts. The first part would be for everyone, presenting the results from the last sprint. The second part would be without stakeholders, talking about technical difficulties. If the stakeholders are included in the second part as well, they might lose interest and skip upcoming meetings.

Consequences
Benefits:

- Getting faster and better feedback by the stakeholders
- Enabling stakeholders to formulate new requirements more accurate

Liabilities:

- If stakeholders don't see the need for this principle, they will see this as additional time requirements

See Also
M-12: Mob-Testing
P-01: Fully transparent agile project.

List of Figures

List of Tables

Bibliography

[1] Standish Group International. The chaos report. *United States of America*, 2015.

[2] Mike Cohen. *Succeeding with Agile*. Addison-Wesley, 2010.

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

[4] Richard Knaster and Dean Leffingwell. *SAFe 4.5 Distilled: Applying the Scaled Agile Framework for Lean Enterprises (2Nd Edition)*. Addison-Wesley Professional, 2nd edition, 2018.

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

[6] Omar A El Sawy, Pernille Kræmmergaard, Henrik Amsinck, and Anders Lerbech Vinther. How lego built the foundations and enterprise capabilities for digital leadership. *MIS Quarterly Executive*, 15(2), 2016.

[7] Henrik Kniberg and Anders Ivarsson. Scaling agile@ spotify. *online], UCVOF, ucvox. files. wordpress. com/2012/11/113617905-scaling-Agile-spotify-11. pdf*, 2012.

[8] Ömer Uludag, 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 (EDOC)*, pages 191–197. IEEE, 2018.

[9] Ömer Uludağ, Nina-Mareike Harders, and Florian Matthes. Documenting recurring concerns and patterns in large-scale agile development. 2019.

[10] Sabine Buckl, Florian Matthes, Alexander W Schneider, and Christian M Schweda. Pattern-based design research–an iterative research method balancing rigor and relevance. In *International Conference on Design Science Research in Information Systems*, pages 73–87. Springer, 2013.

[11] Marian Stoica, Marinela Mircea, and Bogdan Ghilic-Micu. Software development: Agile vs. traditional. *Informatica Economica*, 17(4), 2013.

[12] Ken Schwaber. Scrum development process. In *Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA*, pages 117–134, 1995.

[13] Kent Beck and Cynthia Andres. Extreme programming: Embrace change, 1999.

[14] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Manifesto for agile software development. 2001.

[15] Helen Sharp, Hugh Robinson, Judith Segal, and Dominic Furniss. The role of story cards and the wall in xp teams: a distributed cognition perspective. In *AGILE 2006 (AGILE'06)*, pages 11–pp. IEEE, 2006.

[16] Kent Beck. *Test-driven development: by example.* Addison-Wesley Professional, 2003.

[17] Laurie Williams and Robert Kessler. *Pair programming illuminated.* Addison-Wesley Longman Publishing Co., Inc., 2002.

[18] Scott Ellis. Frameworks, methodologies and processes. `http://vsellis.com/frameworks-methodologies-and-processes/`, 2008. Accessed: 2019-10-13.

[19] Barry W. Boehm. Verifying and validating software requirements and design specifications. *IEEE software*, 1(1):75, 1984.

[20] Kai Petersen, Claes Wohlin, and Dejan Baca. The waterfall model in large-scale development. In *International Conference on Product-Focused Software Process Improvement*, pages 386–400. Springer, 2009.

[21] Ken Schwaber and Jeff Sutherland. The scrum guide-the definitive guide to scrum: The rules of the game. *SCRUM. org, Nov-2017*, 2017.

[22] scrum.org. What is sprint planning? `https://www.scrum.org/resources/what-is-sprint-planning`, 2019. Accessed: 2019-10-30.

[23] scrum.org. What is a daily scrum? `https://www.scrum.org/resources/what-is-a-daily-scrum`, 2019. Accessed: 2019-10-31.

[24] George A Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 101(2):343, 1994.

[25] AOE GmbH. Agile methods & processes in companies. `https://www.aoe.com/en/agile.html`, 2019. Accessed: 2019-11-06.

[26] braintime.de. Safe kompakt. `https://www.braintime.de/methoden/ueberblick-scaled-agile-framework-beratung/safe-grundlagen-kompakt/`, 2019. Accessed: 2019-11-06.

[27] KnowledgeHut. Less vs safe®: Which certification should you choose and why? `https://www.knowledgehut.com/blog/agile/less-vs-safe-which-certification-should-you-choose-and-why`, 2019. Accessed: 2019-11-06.

[28] Alexander M Ernst. *A pattern-based approach to enterprise architecture management*. PhD thesis, Technische Universität München, 2010.

[29] Veli-Pekka Eloranta, Marko Leppänen, and Kai Koskimies. Using domain model for structuring pattern language. In *In: Peltonen, J.(ed.). SPLST'09 & NW-MODE'09, Proceedings of 11th Symposium on Programming Languages and Software Tools and 7th Nordic Workshop on Model Driven Software Engineering, 26-28.8. 2009, Tampere, Finland. Tampere University of Technology. Department of Software System. Report*, 2009.

[30] Amr Elssamadisy. *Agile Adoption Patterns: A Roadmap to Organizational Success (Adobe ebook)*. Addison-Wesley Professional, 2008.

[31] James O Coplien. Software design patterns: Common questions and answers. *The patterns handbook: Techniques, strategies, and applications*, 13:311, 1998.

[32] Christoph Caprano Florian Matthes Ömer Uludağ, Martin Kleehaus. Identifying and structuring challenges in large-scale agile development based on a structured literature review. *Chair for Informatics 19*, 1(1):7, 2017.

[33] Julian M Bass. Artefacts and agile method tailoring in large-scale offshore software development programmes. *Information and Software Technology*, 75:1–16, 2016.

[34] Jack Quinan and Christopher Alexander. A Pattern Language: Towns, Buildings, Construction. *Leonardo*, 14(1):80, 1981. Accessed: 2019-12-12.

[35] Yehuda E. Kalay. *Architecture's new media: Principles, theories, and methods of computer-aided design.* MIT Press, 2004.

[36] M.J. Mahemoff and L.J. Johnston. Principles for a usability-oriented pattern language. In *Proceedings 1998 Australasian Computer Human Interaction Conference. OzCHI'98 (Cat. No.98EX234)*, pages 132–139, Adelaide, SA, Australia, 1998. IEEE Comput. Soc.

[37] Ward Cunningham and Ken Beck. Constructing abstractions for object-oriented applications. *Journal of Object-Oriented Programming*, 2(2):17–19, 1989.

[38] Nikos A. Salingaros. The structure of pattern languages. *arq: Architectural Research Quarterly*, 4(2):149–162, 2000.

[39] Hanna Kallio, Anna-Maija Pietilä, Martin Johnson, and Mari Kangasniemi. Systematic methodological review: developing a framework for a qualitative semi-structured interview guide. *Journal of advanced nursing*, 72(12):2954–2965, 2016.

[40] Katie Moon, Tom Brewer, Stephanie Januchowski-Hartley, Vanessa Adams, and Deborah Blackman. A guideline to improve qualitative social science publishing in ecology and conservation journals. *Ecology and Society*, 21(3), 2016.

[41] INDIANA UNIVERSITY BLOOMINGTON. Internal validity. `http://www.indiana.edu/~p1013447/dictionary/int_val.htm`. Accessed: 2019-12-12.