TECHNISCHE UNIVERSITÄT MÜNCHEN

TUM School of Computation, Information and Technology

Empirical Analysis of the Adoption of Large-Scale Agile Development Methods

Ömer Uludağ

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Prof. Dr. Martin Bichler
Prüfer der Dissertation:	1.	Prof. Dr. Florian Matthes
	2.	Prof. Dr. Stefanie Rinderle-Ma

Die Dissertation wurde am 12.04.2022 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 17.11.2022 angenommen.

Zusammenfassung

Motivation: Seit der Veröffentlichung des Agilen Manifests im Jahr 2001 haben sich agile Methoden zu einer erfolgreichen Vorgehensweise für Softwareprojekte entwickelt, die ständig von externen Faktoren, wie sich ändernden Kundenanforderungen, neuen technologischen Fortschritten und sich ändernden rechtlichen Bedingungen, beeinflusst werden. Angeregt durch die erfolgreiche Anwendung agiler Praktiken in kleinen Projekten, beschäftigen sich viele Softwareexperten mit der Einführung dieser Methoden in großen Projekten und Unternehmen. Mit der steigenden Popularität der Skalierung agiler Methoden ist in den letzten Jahren auch die wissenschaftliche Forschung zu diesem Thema aufgeblüht. Obwohl die Zahl der wissenschaftlichen Artikel in den letzten Jahren zugenommen hat, gibt es noch keine wissenschaftliche Arbeit, die das Wissen über die skalierte agile Softwareentwicklung strukturiert und einen Überblick über den Stand der Forschung in diesem Bereich gibt. Die vorliegende Dissertation bietet die erste systematische Untersuchung des aktuellen Forschungsstandes im Bereich der skalierten agilen Softwareentwicklung. Sie liefert Forschern einen Rückblick auf die vergangenen 13 Jahre der Forschung zu skalierten agilen Softwareentwicklungsmethoden und eine Forschungsagenda für zukünftige Forschungen. Auf der Grundlage dieser Forschungsagenda werden Forschungsthemen aufgegriffen, die in den bisherigen Studien noch nicht ausreichend untersucht wurden: fehlende Übersicht und Analyse von agilen Skalierungsrahmenwerken, fehlende Identifikation und Dokumentation von Mustern sowie mangelnde Untersuchungen in Bezug auf die Zusammenarbeit zwischen Architekten und agilen Teams. Die vorliegende Dissertation kommt dieser Aufforderung durch den Einsatz verschiedener Forschungsmethoden nach.

Forschungsdesign: In dieser Dissertation wurden verschiedene Forschungsmethoden angewandt, um die vier Forschungslücken zu schließen, die sich in drei Kategorien unterteilen lassen: primäre Forschungsmethoden (Umfragen und Fallstudien) sekundäre Forschungsmethoden (systematische Mapping-Studien und strukturierte Literaturrecherchen) und design-basierte Studienmethoden (muster-basierte Designforschungsvorhaben).

Ergebnisse: Die Resultate zeigen, dass die Literatur über 150 Unternehmen berichtet, die agile Methoden skaliert einsetzen, dass die Industrie und Wissenschaft ein wachsendes Interesse an diesem Thema zeigen und dass die Forschung in diesem Bereich in 10 Forschungsstränge unterteilt werden kann. Es wurden 22 verschiedene Skalierungsrahmenwerke mit unterschiedlichen Reifegraden ermittelt, die Standardlösungen zur Verbesserung der Agilität von Unternehmen bieten. Die Ergebnisse verdeutlichen, dass die Interessensgruppen in skalierten agilen Softwareentwicklungsprojekten mit zahlreichen Anliegen konfrontiert sind, die durch die Anwendung verschiedener Muster adressiert werden können. Die Analyse bestätigt, dass die Einbindung von Architekten in skalierte agile Softwareentwicklungsvorhaben von entscheidender Bedeutung ist, um die Vorhaben mit den übergeordneten Geschäftszielen in Einklang zu bringen.

Beitrag: Die vorliegende Dissertation gibt einen systematischen Überblick über die Forschungslandschaft im Bereich der skalierten agilen Softwareentwicklung und stellt eine Forschungsagenda vor, die sowohl von Novizen als auch von erfahrenen Forschern als Ausgangspunkt für ihr zukünftigen Forschungsvorhaben genutzt werden kann. Darüber hinaus wird ein strukturierter Vergleich verschiedener Skalierungsrahmenwerke vorgestellt und die Einführung eines Rahmenwerks

bei einem großen deutschen Automobilhersteller präsentiert, das Praktikern als Entscheidungsgrundlage für die Auswahl von geeigneten Rahmenwerken dienen kann. Außerdem wird eine Liste von Interessensgruppen-bezogenen Anliegen und Mustern vorgestellt, die Praktiker als Leitfaden verwenden können. Schließlich werden Erkenntnissen über die Zusammenarbeit zwischen Architekten und agilen Teams und die erwarteten Verantwortlichkeiten von Architekten aufgeführt, die als Handlungsempfehlungen für Praktiker bei der Entwicklung eines Zusammenarbeitsmodells für Architekten und agile Teams dienen können.

Limitationen: Die enthaltenen Publikationen weisen potenzielle Limitationen hinsichtlich ihrer internen und externen Validitäten und Konstrukt- und Schlussfolgerungsvaliditäten auf. Obwohl die Ergebnisse aus den Publikationen mit Vorsicht zu interpretieren sind, wurden angemessene Gegenmaßnahmen ergriffen, um potenzielle Risiken zu limitieren.

Ausblick: Da die bestehende Forschung von explorativer Evaluationsforschung dominiert wurde, sollten zukünftige Forschungsvorhaben wissenschaftlich fundierte Rahmenwerke, Methoden und Instrumente kreieren, um die Bedürfnisse der Praktiker abzudecken. Da die vorliegende Dissertation einen ersten Versuch unternommen hat, verschiedene Skalierungsrahmenwerke qualitativ zu analysieren, sollten Forscher diese Analyse durch quantitative Instrumente erweitern, um ihre Stärken und Schwächen statistisch zu bewerten. Angesichts des begrenzten Umfangs der vorliegenden Publikationen werden in dieser Dissertation nur neun Muster und Konzepte in ihrer Gesamtheit aufgezeigt. Da die vorgestellten Muster und Konzepte nur einen Teil der gesamten Mustersammlung darstellen, können die bisherigen Ergebnisse durch die Veröffentlichung eines Musterkatalogs erweitert werden. Da der Reifegrad eines Unternehmens in Bezug auf agile Praktiken die Arbeitsweise von Architekten und agilen Teams prägt, sollten zukünftige Forschungsarbeiten mögliche Zusammenhänge zwischen dem agilen Reifegrad eines Unternehmens und den Zusammenarbeitsmodellen zwischen Architekten und agilen Teams aufdecken, insbesondere in Bezug auf die Verteilung von Verantwortlichkeiten für architekturelle Entscheidungen.

Abstract

Motivation: Since the release of the Agile Manifesto in 2001, agile methods have become a successful way in software projects being continuously affected by external determinants, such as changing customer demands, new technological advancements, and shifting regulatory conditions. Inspired by the successful adoption of agile practices in small projects, many software practitioners are engaged in adopting these methods in large projects and companies. With the increasing popularity of scaling agile methods, scientific research on the topic has started to emerge during the past few years. Although the number of scientific articles has increased in recent years, there is still no work structuring the body of knowledge on large-scale agile development and providing a state-of-the-art overview of this research field. This dissertation offers the first systematic exploration of the state of the art in large-scale agile development. It offers researchers a retrospective of the past 13 years of research on the large-scale application of agile methods and a research agenda for future research. Based on this research agenda, we address research themes that have not been adequately investigated in existing studies: lack of overview and analysis of scaling frameworks, missing identification and documentation of patterns, and lack of investigation related to the collaboration between architects and agile teams. We respond to this call through a mix of different research methods.

Research Design: This dissertation used various research methods to address the four research gaps, which can be divided into three categories: primary research methods (surveys and case studies), secondary research methods (systematic mapping studies and structured literature reviews), and design-based study methods (pattern-based design research endeavors).

Results: Our findings show that the literature reports over 150 companies adopting large-scale agile methods, the industry and academia show a growing interest in the topic, and the research in this field can be divided into 10 research streams. We identified 22 scaling frameworks with varying degrees of maturity that provide off-the-shelf solutions for improving the agility of companies. Our results show that stakeholders in large-scale agile development endeavors are confronted with numerous concerns, which can be addressed by the application of various patterns. Our results confirm that the involvement of architects in large-scale agile development efforts is of great importance in aligning the efforts with the overarching enterprise goals.

Contribution: We present a systematic overview of the large-scale agile development research landscape and provide a research agenda that can be used by novices unfamiliar with the research field as well as by experienced researchers as a starting point for their research efforts. We provide a structured comparison of various scaling frameworks and showcase the adoption of a framework in a large German automobile manufacturer that practitioners can use as a decision-making basis for selecting an appropriate framework. We contribute a list of stakeholder-related concerns and patterns that practitioners can use as a reference guide. We present lessons learned regarding the collaboration between architects and agile teams and the anticipated responsibilities of architects that can be used as recommended actions for practitioners to develop a collaboration model for architects and agile teams.

Limitations: The embedded publications show potential limitations related to the internal, external, construct, and conclusion validities. While the findings from the publications should be taken with caution and should be interpreted accordingly, adequate countermeasures have been taken to limit potential threats.

Future Research: As the existing research has been dominated by exploratory evaluation research, future research efforts should create rigorously developed frameworks, methods, and tools to meet the needs of practitioners. Since we made an initial attempt to analyze different scaling frameworks qualitatively, we advise researchers to extend this analysis by quantitative instruments to assess their strengths and weaknesses statistically. Given the limited scope of the embedded publications, this dissertation presents only nine patterns and concepts in their entirety. As the shown patterns and concepts represent a fraction of the entire pattern collection, our results can be extended by publishing a pattern catalog. As the maturity of a company towards agile practices influences the way of working of architects and agile teams, future work should perform explanatory studies to reveal potential correlations between the agile maturity of a company and the collaboration models between architects and agile teams, especially related to the allocation of responsibilities for architecture-related decisions.

Acknowledgment

This dissertation emerged from my work as a research assistant at the Chair for Software Engineering for Business Information System (sebis) at the Technical University of Munich (TUM). I enjoyed a pleasant time with my advisors, colleagues, and students during this period.

First and foremost, I want to express my special appreciation and gratitude to my doctoral father and supervisor, Prof. Dr. Florian Matthes, for allowing me to work on this fascinating research topic under the best possible conditions. I would like to thank him for encouraging my research and allowing me to grow as a research scientist. Further, I want to express my sincere gratitude to Prof. Dr. Stefanie Rinderle-Ma for being the second supervisor of my dissertation.

The sebis chair provided an excellent environment for my research. This dissertation would not have been possible without the support of my colleagues. I would like to thank all my colleagues from the sebis chair for their great support, particularly (in alphabetical order), Gloria Bondel, Ulrich Gallersdörfer, Gonzalo Munilla Garrido, Ingo Glaser, Dr. Matheus Hauder, Dr. Pouya Aleatrati Khosroshahi, Dr. Martin Kleehaus, Dr. Thomas Kugler, Dr. Jörg Landthaler, Nektarios Machner, Sascha Nägele, Pascal Maurice Philipp, Phillip Schneider, and Fatih Yılmaz. Special thanks to our administrative staff, Aline Schmidt and Jian Kong, for supporting me regarding administrative matters, which allowed me to dedicate more time to my research.

This dissertation also profited from a great collaboration with leading researchers in the field of large-scale agile development. Thus, I would like to thank Prof. Dr. Torgeir Dingsøyr for offering me the opportunity to visit him and his research group at SINTEF in Trondheim and his invaluable feedback on several research endeavors. Moreover, I want to express my appreciation for organizing the first-ever lecture on large-scale agile development at TUM. I also would like to express my gratitude to Prof. Dr. Maria Paasivaara, Prof. Dr. Casper Lassenius, and Abheeshta Putta for their warm hospitality at the Aalto University in Helsinki and their valuable feedback in numerous meetings. I want to thank them for their outstanding contributions in jointly published articles in the field of large-scale agile development. I am profoundly grateful to our industry partners that supported our research efforts.

I want to express my thankfulness to the students who wrote their theses under my guidance or supported my research as student assistants.

Finally, I want to thank my family for their support. I want to express my sincere gratitude to my partner Eda Uludağ for her unconditional love and understanding. I also would like to thank my parents, Tercan and Adnan Uludağ, sisters, Özge Altun and Dilara Uludağ, uncle, Ali Taştemir, and brother-in-law, Mehmet Altun, for supporting me throughout my entire life.

Garching b. München, 08.04.2022
Ömer Uludağ

Table of Contents

List of Figures

List of Tables

Part A

CHAPTER 1

Introduction

"Science is the most reliable guide in life."

Mustafa Kemal Atatürk

This dissertation empirically analyzes the large-scale adoption of agile software development methods. It fills several research gaps by exploring the state of the art, analyzing the adoption of scaling agile frameworks, identifying and documenting patterns, and exploring the collaboration between architects and agile teams. Section 1.1 motivates the dissertation by introducing the topic. Section 1.2 presents the research questions that guide this research endeavor. Section 1.3 outlines the structure of the dissertation and summarizes the embedded publications.

1.1. Motivation

Due to accelerating market dynamics, continuous development of new technological advancements, and ever-changing customer demands, contemporary business environments are driven by unpredictability [SKL07, WW15]. Thus, capabilities such as adaptability, flexibility, and learning become critical determinants of a company's success [OBS06, SKL07]. In today's world, software applications have become a centerpiece of many products and services [PW10a]. Over the past four decades, traditional development methods, including waterfall and spiral models, have been widely used for software projects due to their predictability and stability [BT03, FV06, Cho08]. However, these traditional methods also exhibit shortcomings, including slow adaptation to constantly changing business requirements and a tendency to run over budget and behind schedule [Boe02, Cho08]. As traditional methods require extensive planning and documentation, they are often referred to as heavyweight development methods [Boe02]. Traditional methods cannot

respond to changes that constantly confront software development projects, often determining a software product's success or failure [WC03, Ket07]. As traditional methods have increasingly reached their limits, software development methods dramatically evolved from traditional sequential approaches to more flexible and adaptive development approaches [PW10a].

This exigency sparked the agile movement in the 1990s, which led to the development of several agile methods, including Scrum [SM02] and Extreme Programming (XP) [Bec00], and the creation of the Agile Manifesto [BBvB+01] in 2001 [Ket07]. Over the past decades, the software development practice has seen significant growth in the usage of agile methods [HSG18], which have reshaped and brought unprecedented changes to software development practice [MJ10, DNBM12]. Unlike traditional approaches that focus on upfront plans and documentation, agile methods assume that software is adaptively developed based on rapid iterations and frequent feedback loops [NMM05, ASRW17]. Companies aim to realize several benefits by adopting agile practices, including accelerating software delivery, increasing team productivity, improving alignment between business and IT, and improving software quality [Dig21]. Agile methods have also received criticism in the industry [Boe02]. First, they were initially designed for small projects with co-located teams and may not be suitable for large-scale environments [DD09, DMFS18]. Second, agile methods do not rely solely on the appropriate application of individual tools or practices. Instead, they often demand a holistic way of thinking and mindset, necessitating a change in the entire culture [MKK10, KML20]. Despite these known challenges, practitioners have paid much attention to agile methods [DNBM12, ASRW17].

Given the successful application of agile methods in small projects, companies are inspired to apply them in large-scale projects and organizations [DFI14, AR16]. The latest State of Agile Survey [Dig21] reflects this industry trend, stating that over half of their 1,382 respondents work in companies where most teams are agile. Although early advice from the agile community was that scaling agile methods to larger projects and organizations is "*probably the last thing anyone would want to do*" [RME03], and advice from various fields is to decrease the size of software projects as much as possible [And17], companies are still interested in scaling agile methods to larger endeavors [DFP19]. One plausible explanation is that modern solutions often require too much work for a single team or that new solutions are so complex or dependent on existing systems that it is considered inefficient or impractical to divide the development efforts into small projects, making agile methods a way to reduce risk at scale while facilitating innovation [DFP19]. Despite the challenges of adopting agile methods on a large scale, the idea of scaling permeates almost all continents and industries [UPP+22].

In recent years, the increasing popularity of the large-scale application of agile methods in the industry also sparked an academic interest. At the International Conference on Agile Software Development 11 years ago, industry practitioners were asked to list topics they felt should be studied [DMFS18]. They voted "*agile and large projects*" as one of the most burning research questions [FS10]. Since then, nine International Workshops on Large-Scale Agile Development at the annual International Conferences on Agile Software Development have gathered researchers and practitioners to discuss recent studies on large-scale agile development (cf. [DM13, MD17]). Numerous research results have been shared at scientific conferences and journals, and the body of knowledge on this topic has grown (cf. [DPL16, EWC21]).

Aside from the increasing number of primary studies, several secondary studies have also been

published in recent years, most of which are systematic literature reviews [UPP+22]. Existing systematic literature reviews perform in-depth analysis on specific research questions and topics within large-scale agile development, e.g., describing challenges and success factors of large-scale agile transformations or depicting challenges and benefits related to the adoption of scaling agile frameworks. However, none of the identified literature reviews provides an adequate overview of the entire research field [UPP+22]. The number of available reports and findings often proliferates in maturing research areas, making a summary and overview of the research field indispensable [PFMM08]. Consequently, this observation represents a notable research gap within the research field, which we aim to address in this dissertation.

Ever since the first study on large-scale agile development in 2007, researchers worldwide have lavished attention on studying the large-scale application of agile methods. Since then, several research streams have emerged in this field, e.g., agile architecture, agile practices at scale, and scaling agile frameworks [UPP+22]. While extensive work has been done in some research streams, other streams still show significant research gaps [UPP+22]. First, despite the ongoing research on scaling frameworks (cf. [PPL18, CC19b, TSD19]), existing studies lack a comprehensive overview of these frameworks and an analysis of their raison d'être and claimed benefits and challenges of their adoption. Second, compared to the rich body of agile software development literature describing typical challenges (cf. [HBP09, ISM+15]) and best practices (cf. [BDS+99, CH04, BCS+10, Scr21]), the documentation of stakeholder concerns and patterns in large-scale agile development is still scarce. Third, despite the criticisms of agile methods and, in particular, of large-scale agile development for their lack of focus on architecture [DD08, DM14, RWN+15], there is some ambiguity in the literature on the topic of agile architecture in large-scale agile development [UPP+22], primarily related to collaboration between architects and agile teams and responsibilities of architects in such development efforts. Within the scope of the dissertation, we also aim to address these three research gaps.

1.2. Research Questions

In the following, we motivate four research questions that guide this dissertation.

Research Gap 1: *There is a lack of systematic analysis of the body of knowledge and research gaps on large-scale agile development.*

In recent years, large-scale agile development has received much attention from academics and practitioners [DMO18], and research publications on large-scale agile development have been published at scientific conferences and journals [DPL16, MD17]. As a result, the body of knowledge on large-scale agile development has grown [UPP+22]. Although scientific articles on large-scale agile development are mainly primary studies, several secondary studies synthesize knowledge on specific topics related to large-scale agile development [UPP+22]. A total of 13 secondary studies were identified as simple or systematic literature reviews that examine specific research topics [UPP+22], e.g., identifying and describing challenges and success factors in large-scale agile endeavors (cf. [DPL16, SKCK17]). None of them provide an overview of the entire research field, which is why existing secondary studies fail to present the state of the art [UPP+22]. Given that the steadily growing number of primary studies is a valuable indicator of

the increasing maturity of a research field, a critical tipping point has been reached to facilitate the aggregation of the existing literature and organize the research area [UPP+22].

We address this gap by conducting a systematic mapping study that analyzes peer-reviewed publications and provides an overview of the state of the art. We first analyze what is referred to in the literature as "*large-scale agile development*" and then examine publication trends and characteristics of existing research. We also reveal the seminal works in this research area, study the field's main research streams, and create a research agenda for future research endeavors. Against this backdrop, we formulate the first research question as follows:

Research question 1 (RQ1)

What is the state of the art in large-scale agile development research?

Research Gap 2: *Empirical evidence on the adoption of scaling agile frameworks is still limited. The existing literature disregards providing a comprehensive overview of scaling frameworks. There is no analysis of the rationale behind their creation and the benefits and challenges of their adoption from their inventors' perspective. Scientific literature providing in-depth case studies on the adoption of scaling frameworks is still scarce.*

Several scaling frameworks have been created by consultants to address issues associated with adopting agile practices in large-scale organizations and projects [AR16, CC19b]. These frameworks claim to provide off-the-shelf solutions to scaling and incorporate predefined workflow patterns to deal with challenges related to a large number of teams, inter-team coordination, and customer involvement [DMFS18]. As large companies are pressured to become more agile, they have begun to adopt these frameworks at an increasing rate [CC19a]. Despite this growing industrial interest [Dig21], the empirical evidence on the adoption of these frameworks is still in its infancy [CC19a]. While a few studies compare scaling frameworks (cf. [AR16, TSD19]), these tend to focus on the most popular ones, i.e., Scaled Agile Framework (SAFe), Large-Scale Scrum (LeSS), or Disciplined Agile Delivery (DAD), and neglect to analyze the full spectrum of existing frameworks. Although some papers report on the benefits and challenges of adopting these frameworks (cf. [CC19b, EWC21]), they do not provide first-hand information from the frameworks' creators about the key ideas behind the creation of these frameworks and the benefits and challenges that the inventors experienced when adopting their frameworks in organizations. Current case studies primarily focus on SAFe (cf. [Paa17, PPL19]) and the number of studies reporting on the adoption of other frameworks is still scarce.

We fill this gap by conducting a structured literature review that identifies existing scaling frameworks and compares them and surveying their inventors regarding the main reasons behind the creation of their frameworks and claimed benefits and challenges of adopting them. We also provide a case study that describes the adoption of LeSS in four different products of a German car manufacturer. In this light, we articulate the following research question:

Research question 2 (RQ2)

What scaling agile frameworks exist, and what are the reasons, benefits, and challenges of adopting them?

Research Gap 3: *There are no studies describing the challenges stakeholders face in the context of large-scale agile development. Existing literature neglects to provide a structured approach and document best practices to address these challenges.*

Although many companies adopt agile methods at scale to reap their benefits [DPL16], some face unprecedented difficulties, e.g., general resistance to change and coordination issues in multi-team environments [Ket07, DPL16]. Compared to the rich body of agile software development literature describing challenges (cf. [HBP09, ISM+15]) and best practices (cf. [BDS+99, CH04, BCS+10]), the existing literature on large-scale agile development fails to report the challenges stakeholders face in this context and the best practices to address them [UKCM18, UHM19].

Inspired by the *Pattern-based Approach to Enterprise Architecture Management (EAM)* [Ern10, SM15], we address this gap by conducting a structured literature review to identify stakeholders' challenges in large-scale agile development. Following the Pattern-based Design Research (PDR) method [BMSS13a], we first propose the concept of large-scale agile development patterns and then present typical concerns and patterns of agile coaches and scrum masters. Accordingly, we pose the following research question:

Research question 3 (RQ3)

What are the concerns of stakeholders in large-scale agile development, and how can they be addressed?

Research Gap 4: *The collaboration between architects and agile teams in the context of large-scale agile development has hardly been empirically researched yet. The existing literature does not elucidate how architects can support agile teams in this context.*

Agile methods assume that the best architectures emerge from self-organizing teams as a byproduct of daily code design or refactoring activities (emergent design) [Bab09, ABK10]. While emergent design is effective in small projects, it is insufficient in large endeavors since it may cause excessive redesign efforts, architectural divergences, and functional redundancies [LMZ08, Moc09]. An intentional architecture is required that embraces architectural guidelines specifying inter-team design and implementation synchronization [LMZ08, Wat14]. Hence, some degree of architectural planning and governance is vital, as "*agility is enabled by architecture, and vice versa*" [LMZ08, NÖK14]. There are several crucial architectural roles in the context of large-scale agile development: solution architects, who make architectural decisions for agile teams at the team or program level, and enterprise architects, who resolve the technical dependencies of agile teams at the portfolio level and shape the overall strategic architectural vision of the company [HDSD20], whose collaboration with agile teams is of great importance [LMZ08, Wat14]. While there is growing interest from academics and practitioners who want to combine the two concepts of agility and architecture (cf. [BKNÖ14, RWN+15]), empirical studies on investigating the collaboration between architects and agile teams are still limited [HEK15, CCJ+18]. While

a few studies present architectural tactics to support agility at scale (cf. [BNO12, NÖK14]) or discuss integrating EAM practices and agile methods (cf. [HRSM14, HEK15, GvLG21]), they do not observe how solution architects can be involved in large-scale agile development endeavors and how enterprise architects can support agile teams at higher organizational levels.

We fill this gap by providing a case study of a German consumer electronics retailer to describe how solution architects and agile teams collaborate on architectural issues. We also provide a cross-case analysis of five German companies on the expectations of agile teams for enterprise architects and the responsibilities and challenges of enterprise architects in supporting large-scale agile development endeavors. Consequently, we state the following research question:

Research question 4 (RQ4)

How do architects collaborate with agile teams and support them in large-scale agile development?

1.3. Structure of the Dissertation

This dissertation's research endeavor resulted in ten publications to answer the four research questions. The dissertation consists of three parts (see Figure 1.1):

Part A comprises three chapters. Chapter 1 introduces the dissertation by motivating the large-scale agile development research field (see Section 1.1), presenting the four underlying research questions of this dissertation (see Section 1.2), and summarizing the ten publications (see Section 1.3). Chapter 2 recapitulates the theoretical background by providing information on agile and lean software development (see Sections 2.1 and 2.2), large-scale agile development (see Section 2.3), patterns (see Section 2.4), and communication networks (see Section 2.5). These concepts were applied in the embedded publications and might avail the readers of the dissertation. Chapter 3 provides an overview of the dissertation's underlying research design and describes the selected research strategy (see Section 3.1) and methods (see Section 3.2).

Part B comprises the fact sheets of the ten publications organized along with the formulated research questions[1]. While the first publication analyzes existing research on large-scale agile development to provide an overview of the state of the art and identify areas for future research (see P1), the subsequent three publications review existing scaling agile frameworks (see P2 and P3) and investigate the adoption of a popular scaling framework in practice (see P4). Additional three publications focus on identifying challenges of stakeholders in large-scale agile development (see P5), present a conceptual model for documenting patterns (see P6), and describe typical challenges and patterns of agile coaches and scrum masters (see P7). The remaining three publications focus on examining the phenomenon of the collaboration between architects and agile teams in the context of large-scale agile development (see P8, P9, and P10).

Part C consists of three chapters. Chapter 4 discusses the findings of this dissertation by

[1]Nine publications are already published, while one publication is in the first review round (see Part B for further details). References to the ten publications are presented, for example, as see "P1".

discussing the publications' key results (see Section 4.1) and delineating their implications for research and practice (see Sections 4.2 and 4.3). Chapter 5 discusses the limitations of the dissertation before Chapter 6 reflects the initially formulated research questions (see Section 6.1) and outlines potential avenues for future research activities (see Section 6.2).

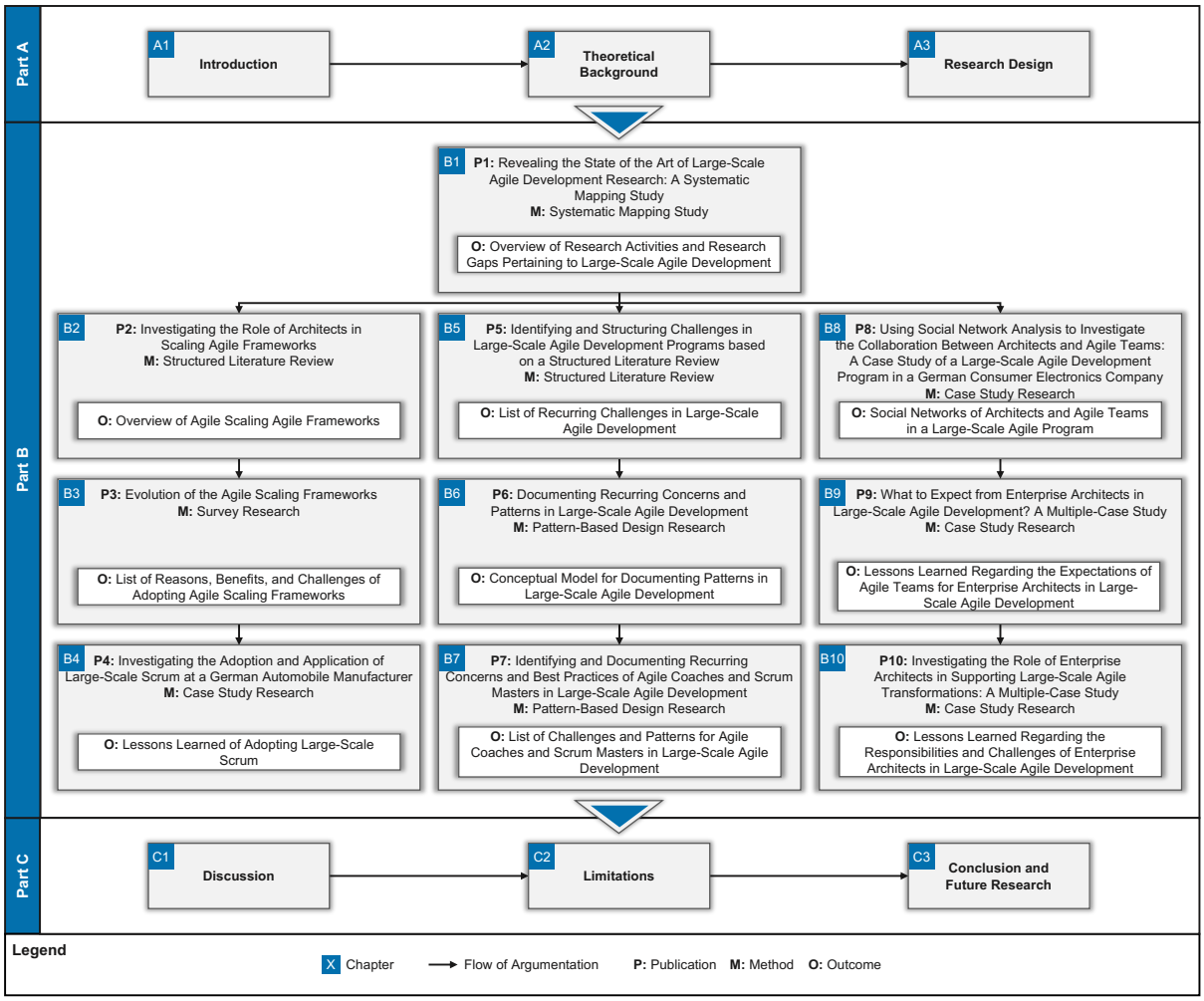


Figure 1.1.: Structure of the dissertation

Table 1.1 outlines the ten publications embedded in Appendix A. For each Publication (P), we highlight its research problem, research design, and main contribution.

P1: Revealing the State of the Art of Large-Scale Agile Development Research: A Systematic Mapping Study Confronted with the imperatives of a rapidly changing world, many companies that develop software by traditional means, characterized by tedious and inflexible development processes, are increasingly reaching their limits. Born from this necessity, agile methods emerged in the 1990s that have transformed the software development practice with a strong emphasis on change tolerance, continuous delivery, and customer involvement. Due to the success of agile methods in small, co-located projects, companies are encouraged to use these methods in larger projects and organizations. Academic interest has also increased

Table 1.1.: Overview of the embedded publications

RQ	No.	Title	Outlet	Type	CORE ranking
RQ1	P1	Revealing the State of the Art of Large-Scale Agile Development Research: A Systematic Mapping Study	JSS 2022*	JNL	A
RQ2	P2	Investigating the Role of Architects in Scaling Agile Frameworks	EDOC 2017*	CON	B
	P3	Evolution of the Agile Scaling Frameworks	AGILE 2021*	CON	B
	P4	Investigating the Adoption and Application of Large-Scale Scrum at a German Automobile Manufacturer	ICGSE 2019*	CON	C
RQ3	P5	Identifying and Structuring Challenges in Large-Scale Agile Development Programs based on a Structured Literature Review	EDOC 2018*	CON	B
	P6	Documenting Recurring Concerns and Patterns in Large-Scale Agile Development	EPLoP 2019*	CON	NR
	P7	Identifying and Documenting Recurring Concerns and Best Practices of Agile Coaches and Scrum Masters in Large-Scale Agile Development	PLoP 2019*	CON	B
RQ4	P8	Using Social Network Analysis to Investigate the Collaboration Between Architects and Agile Teams: A Case Study of a Large-Scale Agile Development Program in a German Consumer Electronics Company	AGILE 2019*	CON	B
	P9	What to Expect from Enterprise Architects in Large-Scale Agile Development? A Multiple-Case Study	AMCIS 2019*	CON	A
	P10	Investigating the Role of Enterprise Architects in Supporting Large-Scale Agile Transformations: A Multiple-Case Study	AMCIS 2020*	CON	A

Outlet:
AGILE: International Conference on Agile Software Development
AMCIS: Americas Conference on Information Systems
EDOC: International Enterprise Distributed Object Computing Conference
EPLoP: European Conference on Pattern Languages of Programs
ICGSE: International Conference on Global Software Engineering
JSS: Journal of Systems and Software
PLOP: International Conference on Pattern Languages of Programs

Type:
CON: Conference
CORE: Computing Research & Education
JNL: Journal
NR: Non-ranked

* All publications are published and peer reviewed, except P1 which is under review.

in recent years with the increasing industrial popularity. Although the body of knowledge has grown, there is still no comprehensive overview of the large-scale agile development research area. This publication [UPP+22] aims to provide the first systematic exploration of the research field by employing a systematic mapping study. This article clarifies the meaning of the term "*large-scale agile development*" by performing an in-depth review of the characteristics of more than 150 cases reported in the publications. This study frames the nature of the research field by assessing publication trends and characteristics of the existing literature. It also explores the intellectual structure of the research area by identifying its seminal works. This study maps the general structure of the research field by identifying and clustering central research themes and outlines a research agenda for future research efforts by analyzing stated research questions in the research streams. Based on this research agenda, the latter nine embedded publications of this dissertation aim to address three research themes that have not been sufficiently investigated yet: the adequate analysis of scaling agile frameworks (see P2, P3, and P4), the identification and documentation of patterns (see P5, P6, and P7), and the appropriate examination of the collaboration between architects and agile teams (see P8, P9, and P10).

P2: Investigating the Role of Architects in Scaling Agile Frameworks. The ideal context for applying agile methods in software projects lies in the "*agile sweet spot*", i.e., small, co-located teams of less than 50 persons with easy access to the user and business experts that develop non-life-critical software. Adopting agile methods outside the sweet spot entails significant challenges to organizations, e.g., coordination issues in multi-team environments. Practitioners have invented several scaling frameworks to address problems associated with the large-scale adoption of agile methods. As large companies face growing pressures and expect to become more agile, the adoption of scaling frameworks has proliferated in the industry. Although there is a body of knowledge on these frameworks, research providing a comprehensive overview of these frameworks is still scarce. Based on a structured literature review, the main objective of this publication [UKXM17] is to present a comprehensive list of scaling agile frameworks and reveal the most mature ones. This paper provides an analysis of 20 scaling frameworks by comparing them with different attributes in terms of descriptive and maturity-related information, e.g., publication date, category, and available training courses and certifications. This study presents an insight into the three most mature scaling frameworks by providing a description of SAFe, LeSS, and DAD. In P3, we extend this analysis by visualizing their evolution and exploring the reasons for their emergence and the benefits and challenges of adopting them.

P3: Evolution of the Agile Scaling Frameworks. The scaling of agile methods poses two significant challenges. First, it entails additional scaling and complexity factors that summon "*bitter spot*" conditions for agile methods. Second, agile methods do not provide sufficient guidance for these scaling and complexity factors. Several scaling frameworks have been proposed to overcome these limitations. Despite their importance in the industry, few empirical studies report the evolution of scaling frameworks, the primary rationale behind their creation, and their adoption's claimed benefits and challenges. Based on surveying their inventors, this publication [UPPM21] builds on the research findings of P2 and aims to sketch the landscape of scaling frameworks and determine their raison d'être. This publication provides an extension of the list of scaling frameworks presented in P1 by expanding the list of identified frameworks by two additional frameworks: Holistic Software Development (HSD) and Parallel Agile (Parallel). This paper visualizes the evolutionary path of scaling agile frameworks by providing a chronological overview of 15 scaling frameworks from 1997 to 2021, showing development start dates, current and intermediate versions, and relationships between frameworks. This study presents the key reasons behind the creation of scaling frameworks by providing a list of 12 reasons, e.g., improving the agility/adaptability of the organization and improving the collaboration of agile teams working on the same product, grouped into four categories: complexity, customer, market, and organization. This article reveals the benefits of adopting scaling frameworks by showing a list of 30 benefits, e.g., enabling frequent product deliveries and enhancing employee satisfaction/motivation/engagement, grouped into two categories: business/product and organization/culture. This paper also reports challenges of adopting these frameworks by depicting a list of 22 challenges, e.g., using frameworks as cooking recipes and using frameworks without understanding for what reasons they should be applied, grouped into three categories: implementation, organization/culture, and scope. While P2 and P3 offer a single-sided view from the existing literature and their methodologists, P4 takes a more practical stance and examines the actual experiences of companies adopting scaling frameworks.

P4: Investigating the Adoption and Application of Large-Scale Scrum at a German

Automobile Manufacturer. Inspired by the success of agile methods at the team level, large companies are scaling agile practices to the organizational level by adopting scaling agile frameworks. Although the number of companies adopting these frameworks is increasing, scientific literature reporting their adoption in the industry is still scarce. Employing a single-case study, the main objective of this publication [UKD+19] is to shed light on the adoption of LeSS at a German automobile manufacturer. This study reports several lessons learned in adopting LeSS by examining the characteristics of its adoption in four different products using seven categories, e.g., start date, duration, and reasons, and juxtaposing its actual implementation with the roles, artifacts, and processes recommended by LeSS.

P5: Identifying and Structuring Challenges in Large-Scale Agile Development Programs based on a Structured Literature Review. Since agile methods were initially designed for small, co-located teams, many companies are uncertain how to introduce them at scale and encounter unprecedented challenges, e.g., coordinating multiple agile teams and dealing with a general resistance to change. Compared to the rich body of agile software development literature describing typical issues associated with the adoption of agile methods, the number of studies reporting challenges observed in large-scale agile development endeavors is still limited. This publication [UKCM18] aims to provide an overview of challenges reported in large-scale agile development through a structured literature review. This study reports on the issues faced by stakeholders in large-scale agile development projects by providing a list of 79 stakeholder-related challenges, e.g., dealing with doubts in people about changes, grouped into 11 challenge categories, e.g., communication and coordination, software architecture, and knowledge management. This paper constitutes the foundation for P6 as it not only provides a proof of concept related to the applicability of a pattern-based approach in large-scale agile development but also an a priori list of pattern candidates.

P6: Documenting Recurring Concerns and Patterns in Large-Scale Agile Development. Compared to the rich body of agile software development literature, the large-scale agile development literature lacks to report on typical concerns stakeholders face and propose best practices for addressing them. Impelled by this research gap, this publication [UHM19] follows the PDR method and strives to propose a pattern language for documenting recurring concerns and patterns of stakeholders in large-scale agile development. This study proposes the concept of large-scale agile development patterns by depicting a conceptual model comprising three different types of patterns, namely coordination, methodology, and viewpoint patterns, and four additional concepts, i.e., stakeholders, concerns, principles, and anti-patterns. This paper demonstrates the applicability of the proposed language by exemplifying the documentation of four patterns, e.g., Strictly Separate Build and Run Stages (principle) and Community of Practice (CoP) (coordination pattern). It also assesses the structure and practical relevance of the proposed pattern language by interviewing 14 experts from ten companies. In P6, we follow the structure of the pattern language and report on typical concerns faced by agile coaches and scrum masters and propose several patterns to address them.

P7: Identifying and Documenting Recurring Concerns and Best Practices of Agile Coaches and Scrum Masters in Large-Scale Agile Development. As one of the key actors in the agile transformation of organizations, agile coaches and scrum masters face numerous challenges, e.g., establishing an agile culture across the organization and fostering informa-

tion sharing. Despite their importance, the existing literature still lacks reporting their typical concerns and proposing patterns to assist them. Following the PDR method, this publication [UM19] strives to reveal typical concerns and patterns of agile coaches and scrum masters. Based on 13 interviews, this study contributes an overview of recurring concerns of agile coaches and scrum masters by providing a list of 57 recurring concerns, 36 of which were already identified by P6 and 21 of which were newly reported by the interviewees. This publication proposes several practices to address the identified concerns by recommending a list of 76 pattern candidates and 15 patterns, of which five are showcased.

P8: Using Social Network Analysis to Investigate the Collaboration Between Architects and Agile Teams: A Case Study of a Large-Scale Agile Development Program in a German Consumer Electronics Company. Scaling agile methods causes not only managerial challenges, e.g., establishing an enterprise-wide agile mindset, but also architecture-related deficiencies, e.g., the negligence of agile practices in assisting software architecting. Propelled by these challenges, academics and practitioners have a growing interest in combining the two concepts of agility and architecture, as some architectural planning and governance is crucial for large-scale agile development endeavors. There is still some ambiguity about the role of architects in these efforts, so researchers have expressed several research needs related to the investigation of collaboration between architects and agile teams. Utilizing a single-case study and social network analysis, the main aim of this publication [UKEM19] is to explore the collaboration between architects and agile teams in a large-scale agile development program of a German consumer electronics company. Based on observing two planning events and conducting seven semi-structured interviews and an online survey with 32 participants, this study contributes a set of social network graphs delineating the architecture-related information exchange of eight agile teams and between solution architects and team members using social network metrics and common communication network patterns. While P8 focuses on describing the collaboration between solution architects and agile teams at the program level, P9 builds on the findings of P8 and takes a more holistic stance by depicting the expectations of agile teams for enterprise architects in supporting large-scale agile development efforts at the enterprise level.

P9: What to Expect from Enterprise Architects in Large-Scale Agile Development? A Multiple-Case Study. In large-scale agile development, the alignment of agile teams becomes a key determinant for achieving desirable organization-wide effects. A crucial vehicle to attain this objective is to establish effective EAM functions that provide a shared target vision across the organization. Until now, many enterprise architects primarily focused on top-down governance, exhibiting an enforcement-centric view that conflicts with the culture and mindset of agile teams, partially driven by the lack of shared expectations between these two groups. Despite this challenge, the existing literature lacks to provide a set of recommendations for overcoming this issue. Employing a multiple-case study in five German companies, the main objective of this publication [UKRM19] is to provide clarity on what expectations agile teams have for enterprise architects. This study contributes a set of lessons learned related to the expectation of agile teams for enterprise architects using the organization-specific agile EAM practice model alongside its seven dimensions, e.g., communication, involvement, and support. This publication highlights the extent to which the expectations of agile teams for enterprise architects are met by triangulating the viewpoints of different stakeholder groups and contrasting the self-perception of enterprise architects with the external perception of agile teams and

managers. P10 represents a continuation of P9 and describes the role of enterprise architects in supporting large-scale agile development endeavors.

P10: Investigating the Role of Enterprise Architects in Supporting Large-Scale Agile Transformations: A Multiple-Case Study. Although the EAM function has established itself as a valuable governance mechanism to coordinate multiple large-scale agile development endeavors, empirical studies still lack evidence on how enterprise architects can support these efforts. Through a multiple-case study of five German companies, the primary goal of this publication [UM20a] is to provide an insight into the enterprise architects' responsibilities in supporting agile teams and the challenges they face when working with agile teams. Based on 16 semi-structured interviews, this article reveals typical tasks enterprise architects perform when supporting large-scale agile development endeavors by reporting a list of 17 responsibilities of enterprise architects, e.g., ensuring the reuse of enterprise assets and fostering technical excellence. This study describes the problems enterprise architects face in supporting agile teams by providing a list of 17 recurring reported challenges, e.g., balancing short-term and long-term planning, dealing with the loss of decision-making power, and managing technical debts.

In addition to the ten publications embedded in this dissertation, we wrote eight additional articles (in-)directly related to the research questions (see Table 1.2). These publications – partly led by co-authors – present additions to the topics discussed in the embedded publications.

Table 1.2.: Overview of the additional publications

RQ	No.	Title	Outlet	Type	CORE ranking
RQ2	P11	Benefits and Challenges of Adopting SAFe – An Empirical Survey	AGILE 2021*	CON	B
	P12	Why Do Organizations Adopt Agile Scaling Frameworks? A Survey of Practitioners	ESEM 2021*	CON	A
RQ3	P13	Large-Scale Agile Development Patterns for Enterprise and Solution Architects	EPLoP 2020*	CON	NR
RQ4	P14	Supporting Large-Scale Agile Development with Domain-Driven Design	AGILE 2018*	CON	B
	P15	Improving the Collaboration Between Enterprise Architects and Agile Teams: A Multiple-Case Study	ADT 2021*	BCH	NR
	P16	Investigating the Establishment of Architecture Principles for Supporting Large-Scale Agile Transformations	EDOC 2019*	CON	B
	P17	Establishing Architecture Guidelines in Large-Scale Agile Development Through Institutional Pressures	AMCIS 2019*	CON	A
	P18	A Tool Supporting Architecture Principles and Guidelines in Large-Scale Agile Development	ADT 2021*	BCH	NR

Outlet:
ADT: Architecting the Digital Transformation
AGILE: International Conference on Agile Software Development
AMCIS: Americas Conference on Information Systems
EDOC: International Enterprise Distributed Object Computing Conference
EPLoP: European Conference on Pattern Languages of Programs
ESEM: International Symposium on Empirical Software Engineering and Measurement

Type:
BCH: Book chapter
CON: Conference
CORE: Computing Research & Education
NR: Non-ranked

* All publications are published and peer reviewed.

Related to RQ2, this study [PUPH21] examines the state of practice in adopting SAFe based on a survey of practitioners worldwide on their perceptions of the benefits and challenges associated

with the adoption. Based on the same survey, this study [PUH+21] investigates the reasons, the expected benefits, and the satisfaction level of companies with adopting scaling frameworks.

Related to RQ3 and similar to P7, this publication [UM20b] follows the PDR method and presents concerns and patterns of solution and enterprise architects.

Related to RQ4, this study [UHK+18] proposes a framework that incorporates components of LeSS and Domain-Driven Design to demonstrate how companies can extend large-scale agile development efforts by agile architecting activities. Based on the same multiple-case study of P9 and P10, this publication [URM21] presents five tactics that enterprise architects can use to enhance their collaboration with agile teams. This study [UPM19] uses the same multiple-case study data as P9 and P10 to reveal companies' current practices and challenges in applying architectural principles to support large-scale agile development efforts. With a similar scope, this publication [UNH19] suggests a collaborative approach for agile teams and enterprise architects to establish and manage architecture principles in large-scale agile development endeavors. This study [UNHM21] extends the approach presented in [UNH19] by proposing a prototypical web application called "*Architecture Belt*" that assists the establishment of architecture principles.

Although these additional publications provide further insights into the research questions, we selected the publications embedded in this dissertation (P1–P10) as the main building blocks.

CHAPTER 2

Theoretical Background

In the theoretical background, we start with a description of the fundamental concepts on which the dissertation and all embedded publications are built, namely agile and lean software development (see Sections 2.1 and 2.2) and large-scale agile development (see Section 2.3). In addition, we introduce the idea of patterns in Section 2.4, which forms the basis for P5, P6, and P7. We present the notion of communication networks in Section 2.5, on which P8 builds to describe the collaboration between architects and agile teams.

2.1. Agile Software Development

In the mid-1980s, Takeuchi and Nonaka [TN86] already stated that a sequential phases approach to product development is not well suited due to the lack of flexibility [STE17]. Similarly, linear and sequential methods, known as the "*Waterfall Model*" [Ben83], were pervasive in software development and generally lacked effectiveness in responding to customer needs, managing changing project scope, and meeting delivery deadlines and costs [FSS02, LGJO17]. In these methods, work begins with the extensive elicitation and documentation of a complete set of requirements, followed by an architectural and high-level design, often delaying the starting point of implementation [CLC04, NÖS12]. In the mid-1990s, some practitioners found these initial development steps frustrating and even impossible [HH02]. The industry and technology were evolving too quickly, requirements were changing at a pace that overwhelmed traditional methods, and customers were increasingly unable to articulate their needs in advance while expecting more from their software [CLC04]. As a result, several consultants independently developed iterative and incremental methods to respond to the inevitable changes they were experiencing [CLC04]. The concept of "*agility*" for executing and organizing software development projects emerged based on ideas from new product development (cf. [TN86]) [SH15]. Hence, agile – de-

noting "*the quality of being agile, readiness for motion, nimbleness, activity, dexterity in motion*" [AWSR03] – software development can be seen as a reaction to traditional ways of developing software and acknowledges the "*need for an alternative to documentation driven, heavyweight software development processes*" [Boe02, CLC04]. The resulting trend was global, and more than 14 lightweight methods were proposed, which later became known as agile methods after the creation of the Agile Manifesto [BBvB+01, CS16]. In 2001, various agile proponents came together to discuss their practices and find common ground, resulting in the formulation of the Agile Manifesto, which still binds agile methods with a common set of values [BBvB+01].

While traditional software development builds on the fundamental assumption that information systems are fully specifiable and are created through meticulous and extensive planning, agile software development assumes that information systems can be created through continuous design, improvement, and testing based on rapid feedback and changes [NMM05]. Hence, agile software development views change as a persistent constant by underlining the continued readiness "*to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while contributing to perceived customer value*" [Con09]. Furthermore, agile software development differs from traditional software development in that it explicitly questions the need for documentation and the use of methods for the sake of methods [CS16]. Table 2.1 provides a comparative summary of traditional and agile software development.

Table 2.1.: Traditional versus agile software development [NMM05]

Characteristic	Traditional development	Agile development
Fundamental assumption	Systems are fully specifiable, predictable, and are built through meticulous and extensive planning	High-quality adaptive software is developed by small teams using the principles of continuous design improvement and testing based on rapid feedback and change
Management style	Command and control	Leadership and collaboration
Knowledge management	Explicit	Tacit
Communication	Formal	Informal
Development model	Life-cycle model (waterfall, spiral or some variation)	Evolutionary-delivery model
Desired organizational form / structure	Mechanistic (bureaucratic with high formalization), aimed at large organizations	Organic (flexible and participative encouraging cooperative social action), aimed at small and medium-sized organizations
Quality control	Heavy planning and strict control, late, heavy testing	Continuous control of requirements, design, and solutions, continuous testing

Although agile software development brought a paradigm shift in software engineering [Raj06], agile development has also evoked a substantial amount of criticism and debates by some practitioners and academics [AWSR03, DD08]:

- Agile development is not new; such practices have been around in software development since the 1960s [MRTR05].
- The lack of focus on architecture leads to sub-optimal design decisions [McB03, RS08].

- There is scant academic evidence for many of the claims of the agile community [McB03].
- XP practices are rarely applicable [Kee02].
- Agile development is more appropriate for small teams, but other practices are better for larger projects [CLC04].

2.1.1. Agile Manifesto

"*[A] bigger gathering of organizational anarchists would be hard to find*", Beck stated [BBvB+01] when seventeen representatives of different agile methods and other sympathizers came together in Utah in early 2001 to discuss the need for new software development methods as an alternative to heavyweight software development processes [CLC04]. "*What emerged was the Agile 'Software Development' Manifesto*" as a result of this gathering [BBvB+01]. Since its articulation in 2001, the Agile Manifesto has become an essential part of the agile movement. It has brought unprecedented changes to software engineering in that it characterizes the binding values of agile methods and how they differ from traditional methods [CLC04, DNBM12]. The Agile Manifesto sets in place four core values and elaborates 12 supporting principles encapsulating the ideas underlying agile software development methods [CS16] (see Figure 2.1).

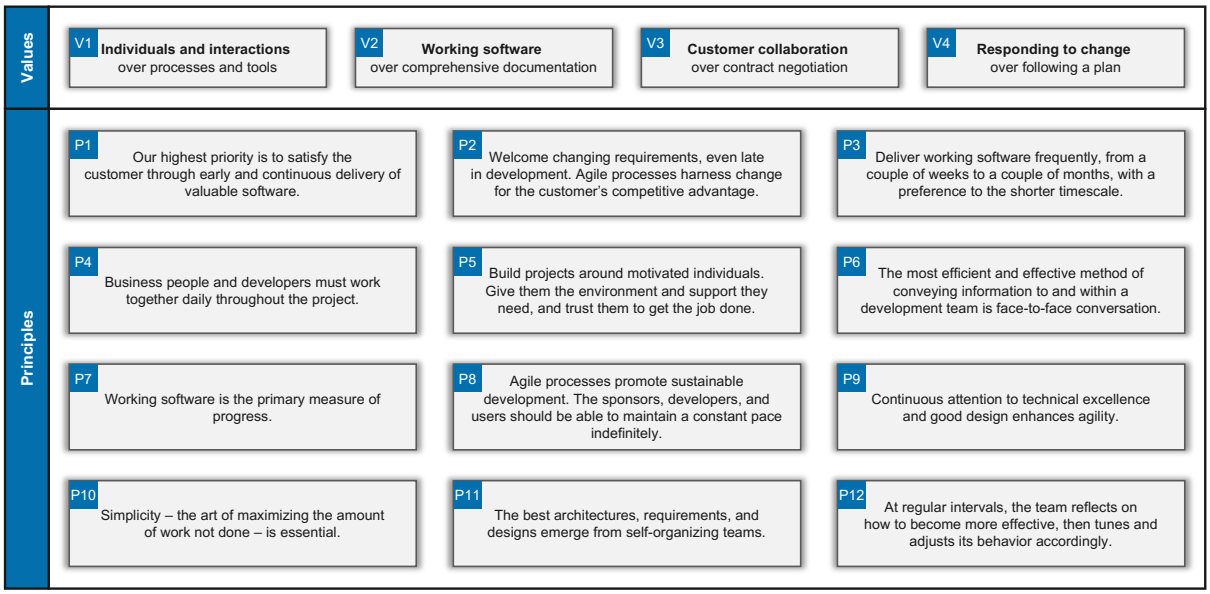


Figure 2.1.: Values and principles of the Agile Manifesto [BBvB+01]

2.1.2. Agile Software Development Methods

Methods for agile software development constitute a collection of different techniques or practices that experienced practitioners have created [CLC04, ÅF06]. These methods share common values and principles, e.g., many are based on iterative and incremental development and best suit collocated teams of about 50 people or fewer who are developing non-life-critical projects

[WC03, CLC04]. At the heart of these methods is the idea of self-organizing teams whose members are collocated and work at a pace that fosters their creativity and productivity. They accommodate changes in requirements at any stage of the development process and ensure that customers are actively involved in the development process [DNBM12].

Various agile methods have been introduced over the past few decades [AWSR03]. Larman and Basili [LB03] refer to the Dynamic Systems Development Method as the first agile method, followed by XP, whose introduction is widely recognized as the starting point for the various agile software development approaches [AWSR03]. Several other methods, adhering to varying degrees to the tenets of the Agile Manifesto, have since followed and appeared on the landscape, including Adaptive Software Development, Crystal Methods, and Feature-Driven Development, to name a few [AWSR03, DD08, DNBM12]. The emergence of new agile methods has exploded at the turn of the 21st century [AWSR03] (see Figure 2.2).

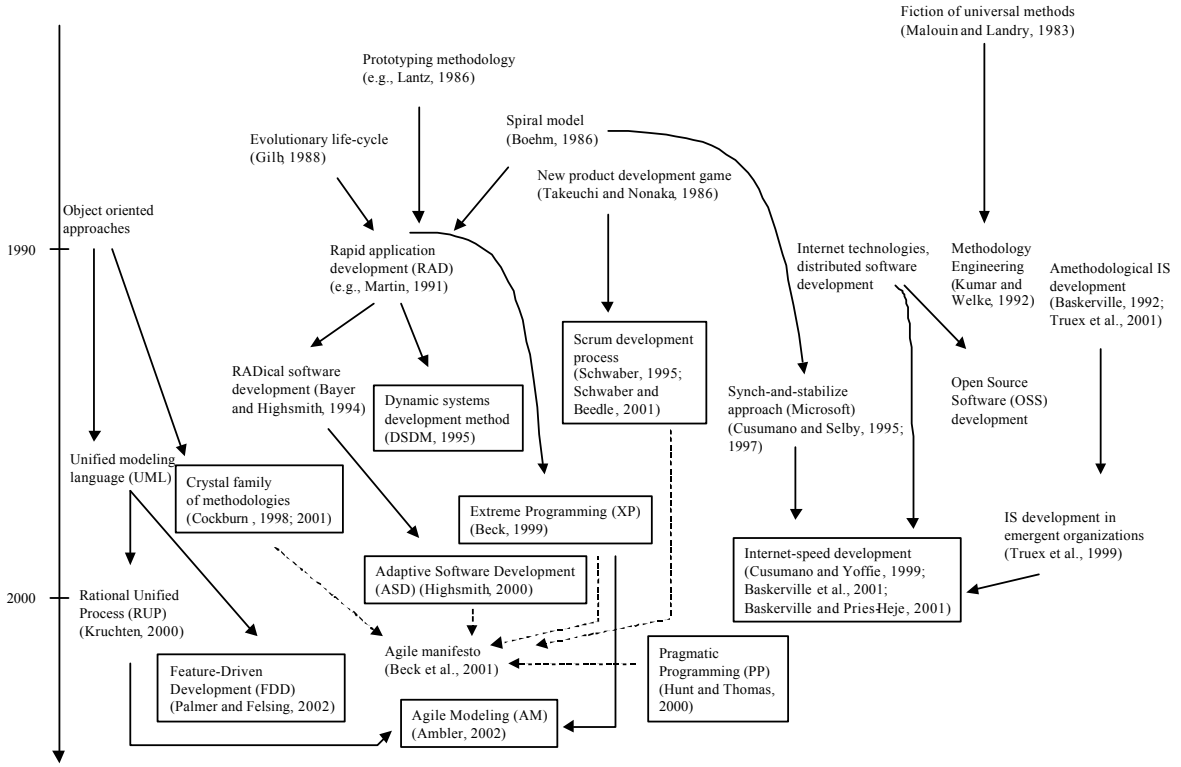


Figure 2.2.: Evolutionary map of agile methods [AWSR03]

Currently, the most widely adopted agile methods in industry are Scrum and XP [HA13], which is why both are detailed below.

2.1.3. Scrum

The "*Scrum*" metaphor has its origins in new product development, initially coined by Takeuchi and Nonaka [TN86] in 1986, describing an innovative approach called the "*Rugby Approach*",

to manage commercial product development [FSOO13, Coo16]. The approach proposed by Takeuchi and Nonaka [TN86] originally stems from rugby, where "*Scrummage*" (or in short "*Scrum*") refers to a formation of players packing closely together with their heads down and attempting to gain possession of the ball [Coo16, CS16]. In product development, a Scrum is a meeting of a self-organizing team to plan its next moves [TN86, SM02]. Intrigued by the idea of [TN86] having a self-organized team in new product development [SVBP07], Jeff Sutherland and Ken Schwaber jointly developed a software development process called "*Scrum*" and introduced it to the public at the OOPSLA Business Object Design and Implementation Workshop in 1995 [Sch95]. Scrum assumes that analysis, design, and development processes are unpredictable due to changing requirements and business demands. Hence, Scrum constitutes an iterative and incremental process for building software in unpredictable environments [Sch95, RJ00].

Figure 2.3 provides a visual representation of the Scrum process, consisting of four core components: values, roles, events, and artifacts (see Table 2.2), described in the following.

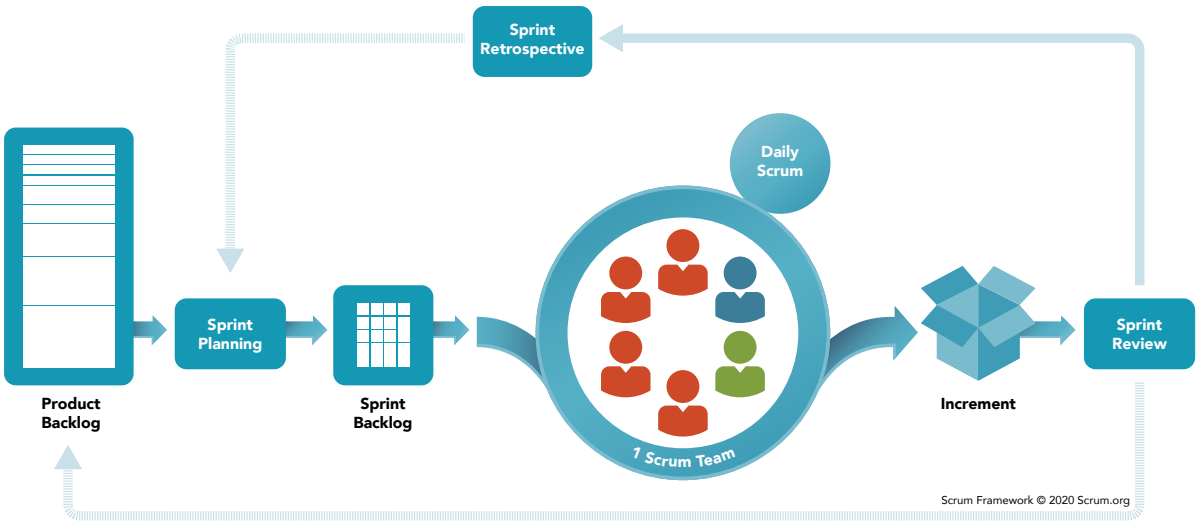


Figure 2.3.: Overview of the Scrum framework [Scr22]

Table 2.2.: Core components of Scrum [SS20]

Scrum values	Scrum roles	Scrum events	Scrum artifacts
• Commitment • Focus • Openness • Respect • Courage	• Scrum team • Developers • Product owner • Scrum master	• Sprint • Sprint planning • Daily scrum • Sprint review • Sprint retrospective	• Product backlog • Sprint backlog • Increment

Scrum values. Scrum formulates five values that give a scrum team direction for its work, actions, and behavior. *Commitment* refers to a team's commitment to achieving its goals. *Focus* pertains to a team's primary focus being on the sprint's work to drive the best possible progress toward these goals. *Openness* signifies a team and its stakeholders being open about the work

and the challenges. *Respect* relates to the team's members respecting each other. *Courage* means that the team's members dare to do the right thing and work on complex problems [SS20].

Scrum roles. A small team of people is the fundamental unit of Scrum. A *Scrum Team* is typically cross-functional, meaning the team members have all the skills needed to develop software in each sprint, and is self-managing, meaning it decides internally who does what, when, and how [SS20]. A scrum team typically consists of 7 ± 2 team members oriented to the number of objects an average person can hold in short-term memory [Mil56]. The scrum team is responsible for all product-related activities and creating a valuable increment in each sprint [SS20]. Scrum defines three distinct roles within a scrum team. First, *Developers* are the people in the scrum team who do the work specified in the sprint backlog [Rub12]. They are committed to creating an increment in each sprint [SS20]. Second, the *Product Owner* ensures that the customer's needs are understood and maintains the product backlog. The product owner prioritizes and clarifies the backlog items, communicating to the developers what needs to be built and ensuring that the project is economically viable [Rub12, SS20]. Third, the *Scrum Master* is a servant-leader for the scrum team and is responsible for the proper execution of Scrum. The scrum master accomplishes this by helping everyone in the team and organization understand the theory and practice behind the framework [SS20].

Scrum events. In Scrum, an iteration is called a *Sprint*, a container for all other Scrum events. A sprint is a time-boxed event, typically a month or less, in which all the work required to create a potentially shippable product increment is performed [Rub12, SS20]. A new sprint begins immediately after the completion of the previous sprint [SS20]. A *Sprint Planning* is a time-boxed meeting, eight hours maximum for a one-month sprint, that initiates the sprint by determining the work for the sprint. The entire scrum team attends the sprint planning meeting and collectively determines what should be part of the sprint backlog for the next product increment [Rub12, SS20]. Every day during a sprint, the developers gather for a short *Daily Scrum* meeting that lasts 15 minutes. The team inspects the progress toward the sprint goal and adjusts the sprint backlog as necessary by adjusting the work plan for the upcoming day. A *Sprint Review* meeting is used after a sprint to review the outcome of the sprint and determine future adaptations. Thereby, the scrum team presents the results of its work to key stakeholders. A sprint review meeting is time-boxed to four hours for a one-month sprint [SS20]. Between the sprint review meeting and the next sprint planning meeting, the entire scrum team holds a *Sprint Retrospective* meeting, which is time-boxed to three hours for a one-month sprint. The goal of the sprint retrospective is to analyze the scrum team's workflows and processes to identify opportunities for improving work and collaboration [Rub12, SS20].

Scrum artifacts. Scrum defines three artifacts [SS20]. First, the *Product Backlog* is the first artifact created for development [Rub12]. It is an emergent, prioritized list of all features of the product. The product backlog is the sole source for the scrum team's work [SS20]. Second, the *Sprint Backlog* is an artifact that contains product backlog items that the scrum team is working on during the current sprint. It consists of the sprint goal, the product backlog items selected for the sprint, and an actionable plan for delivering the increment. Third, an *Increment* is an output artifact created by developers after a sprint. It results from all sprint backlog items completed in the current sprint. Each increment is additive to all previous increments and is thoroughly verified to ensure that all increments function together [SS20].

2.1.4. Extreme Programming

XP is a method that helped agile software development gain public attention, being a dominant methodology in the late 1990s through 2000 before the rise of Scrum [Fow13]. XP was developed by Kent Beck in 1996 while working on a C3 payroll project [ABB98] and further popularized by Beck's book entitled "*Extreme Programming Explained: Embrace Change*" in 1999 [Bec00]. XP was created as a response to the problems caused by the long development cycles of traditional models and owes much of its popularity to developers who were disenchanted with conventional methods and were looking for something new, something "*extreme*" [Bec00, HH02, CLC04]. XP initially started as a "*simple way to get the job done*" [Hau01] using practices that had been proven effective in software development processes [Bec00]. After a series of successful trials in practice [ABB98], XP was theorized, and its fundamental principles and practices were described. The term "*extreme*" comes from the exaggeration of these commonsense principles and practices [Bec00]. Beck [Bec00] defines XP as a "*software development discipline that organizes people to create high-quality software in a more productive manner*". XP is considered an agile method as it is organized into multiple short development cycles to reduce the cost of changes made to accommodate the requirements expressed by customers. XP focuses on the development aspects at the expense of the management aspects and is designed to be fully or partially customizable within an organization. The working teams in XP are small and aim to develop software in an environment where requirements frequently change [GJJG11]. The life-cycle of XP consists of six phases [Bec00] (see Figure 2.4), detailed in the following.

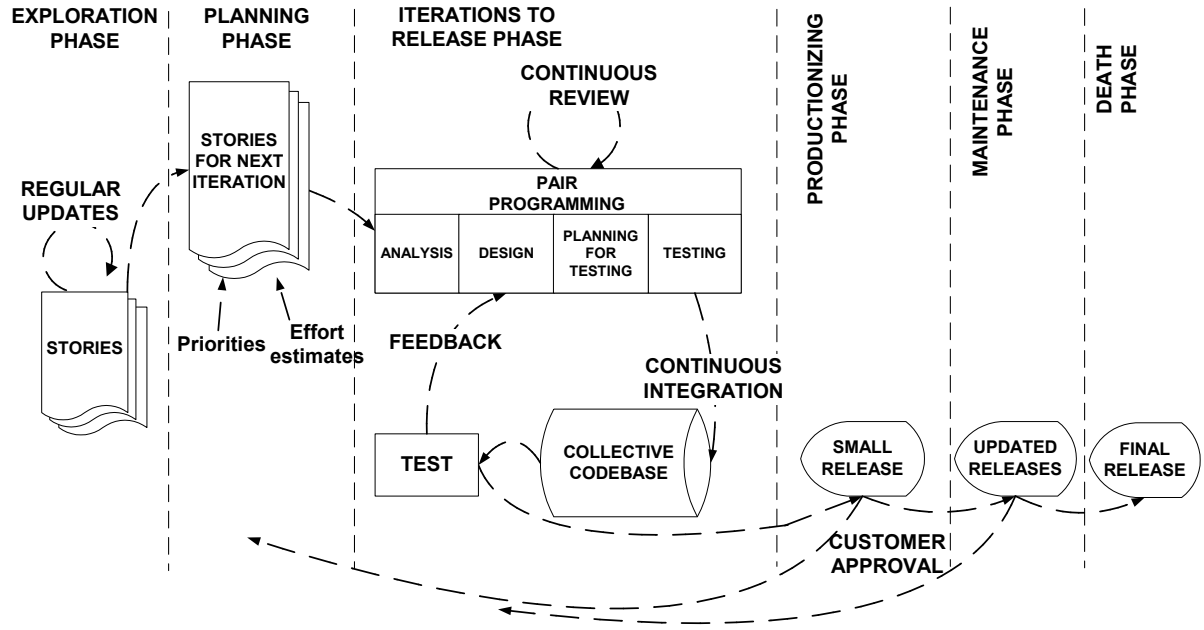


Figure 2.4.: Overview of the XP life-cycle process [Wel01]

XP phases. In the *Exploration Phase*, the customer writes the user stories to describe the features he/she wishes to have in the system's first release. In parallel, the project team familiarizes itself with the tools, technology, and practices it will use. The technology to be used is

tested, and architectural possibilities for the system are explored by building a prototype. The exploration phase lasts from a few weeks to a few months. The customer determines the priority order for user stories in the *Planning Phase*. The programmers estimate the effort required to implement each user story, followed by an agreement on the content of the first release. While the period for the schedule of the first release does not exceed two months, the phase itself takes a few days. The *Iterations to Release Phase* comprises several iterations before the system's first release. The schedule set in the planning phase is decomposed into several iterations that take one to four weeks to implement. In the first iteration, an architecture of the whole system is created. The customer decides which stories are selected for each iteration. The functional tests specified by the customer are performed at the end of each iteration. The system is ready for production at the end of the last iteration. The *Productionizing Phase* comprises additional testing and verification of the system's performance before it is handed over to the customer. In the *Maintenance Phase*, the project runs the system in production while developing new features. The *Death Phase* begins when the customer no longer has stories to implement, when the system is not delivering the desired results, or when it becomes too expensive for further development. The necessary documentation for the system is written in this phase, as no more changes are made to the architecture, design, or code [Bec00, ASRW17].

XP comprises four core components [Bec00]: values, principles, practices, and roles (see Table 2.3), outlined below.

Table 2.3.: Core components of XP [Bec00]

XP values	XP principles	XP practices	XP roles
• Communication • Simplicity • Feedback • Courage • Respect	• Rapid feedback • Assume simplicity • Incremental change • Embracing change • Quality work	• Planning game • Small releases • Metaphor • Simple design • Testing • Refactoring • Pair programming • Collective ownership • Continuous integration • 40-hours work week • On-site customer • Coding standards	• Programmer • Customer • Tester • Tracker • Coach • Consultant • Manager

XP values. There are five XP values [Bec00]. *Communication* fosters active and continuous communication among team members instead of documentation [AASW17]. *Simplicity* pertains to keeping things simple, e.g., creating a simple plan, design, code, or solution. *Feedback* relates to regular feedback spanning various time scales from seconds to months. *Courage* signifies that XP practices require courage, e.g., refactoring a code completed after great effort. *Respect* stands for the importance of self-respect and respect for other members [Bec00, AASW17].

XP principles. In tandem with XP values, XP defines five principles. *Rapid Feedback* implies getting feedback as often as possible [Bec00]. *Assume Simplicity* means treating every problem simple until proven otherwise. *Incremental Change* indicates decomposing each change into a series of small steps. *Embracing Change* refers to accepting changes in design, project, or plan. *Quality Work* suggests that the team produces a valuable product [Bec00, HOV05].

XP practices. XP proposes a set of practices to improve productivity while maintaining quality [Bec00]. The *Planning Game* represents the meeting between the customer and programmers and takes place once per iteration. It consists of two steps. First, programmers estimate the effort required to implement the user stories. Then, the customer decides on the scope and timing of the releases. *Small Releases* encourage programmers to create small and workable parts of the system for the customer in a short time and even release them daily, or at least monthly. Instead of traditional architecture, XP uses the *Metaphor* as a simple shared vision that describes how the system works. *Simple Design* recommends programmers design the most straightforward solution that is implementable at the moment. In XP, *Testing* is accomplished by continuously implementing and running unit tests. It also foresees that the customer writes test plans and functional tests before implementation. *Refactoring* envisages programmers restructuring the system at the end of each iteration by removing duplicates, improving communication, simplifying, and adding flexibility. *Pair Programming* involves two programmers working side-by-side on the same code on a single computer. The *Collective Ownership* states that the code produced does not belong to a specific person but to everyone on the team, which is why anyone can change any part of the code. *Continuous Integration* envisions that a new piece of code is integrated into the code base as soon as it is ready. A *40-Hour Week* limits the working hour for programmers to a maximum of 40 hours per week. The *On-site Customer* requires the customer to be present and available to the team full-time and sit with programmers to answer questions, resolve disputes, and set priorities. *Coding Standards* recommend programmers specify coding rules that make the code easier to understand [Bec00, ASRW17].

XP roles. XP defines seven roles for different tasks and purposes [Bec00]. The *Programmer* is responsible for the project's main deliverable and writes the source code and tests for the system under development. Since there are no analysts, designers, or architects in an XP team, the programmer must perform all of these tasks. The *Customer* creates the stories and functional tests and decides when a requirement is met. The customer sets the priority for the implementation of the requirements. The *Tester* assists the customer in creating functional tests, regularly running the functional tests, broadcasting the test results, and maintaining the testing tools. The *Tracker* monitors the project's metrics and evaluates whether the goal can be achieved within the given time constraints or whether changes in the process are needed. The *Coach* is responsible for the development process and guides the other team members to follow the XP process. The *Consultant* is an external member with extra expertise hired by the team temporarily when the team needs support. The consultant shares his or her knowledge so that the team can solve the problem on its own. The *Manager* is the project coordinator and provides the necessary resources, equipment, and tools for the XP project [Bec00, ASRW17].

2.2. Lean Software Development

"*Lean Thinking*" was born as part of the industrial renaissance in Japan after the Second World War [RMO+19]. Its origins are traced back to the Toyota Production System (TPS) in the 1950s [Ohn88, Lik03], which did not significantly impact the mainstream literature until researchers from the Massachusetts Institute of Technology (MIT) began research under the International Motor Vehicle Program [RMO+19]. Intending to study the differences among the world's au-

tomobile industries, researchers discovered that Japan's automobile industry was far ahead of the American one. By carefully studying Japanese methods, particularly those of Toyota, they conceived an entirely different production system [WJR90]. At that time, Toyota was widely regarded as the most efficient and highest quality manufacturer of motor vehicles in the world [MJ11]. The term "*lean*" refers to the fact that Toyota required less space, workforce, materials, and time to manufacture its products than its Western competitors [WJ97]. According to MIT researchers, lean thinking is about "*doing more with less*" by ideally producing "*the right things, at the right time and in the right place*" [WJR90]. Lean thinking is a school of thought that empowers companies to "*specify value, line up value-creating actions in the best sequence, conduct these activities without interruption whenever someone requests them, and perform them more and more effectively*" [WJ97]. Based on the fundamental principles of industrial engineering, lean thinking is grounded in the philosophy of maximizing value and minimizing waste that is guided by five interrelated key traits [WCC12, RMO+19]:

- *Value* is defined as everything a customer is willing to pay for.
- *Value Stream* is an optimized end-to-end collection of actions required to take a product from customer order to customer support.
- *Flow* means that production activities are organized as a continuous flow that eliminates discontinuities.
- *Pull* indicates that products are produced only when they are needed.
- *Perfection*, or "*Kaizen*" in Japanese, refers to pursuing perfection by continuously identifying and eliminating waste.

TPS is a form of what Womack et al. [WJ97] coined as "*Lean Manufacturing*" [WJR90]. Lean manufacturing can be summarized under two headings: *the removal of waste* and *continuous flow* [PW11]. Lean thinking divides work into value-adding, required non-value-adding, and non-value-adding activities [WJ97]. Non-value-adding activities should be reduced or eliminated as they are considered waste and are referred to as "*Muda*" (garbage in Japanese). Other types of waste include over-production, over-processing or incorrect processing, excess inventories, or unused employee creativity, to name a few [Ohn88, Lik03]. Lean thinking enables to achieve a smooth flow of inventory through the production process [PW10b]. By linking disjointed operations, quality problems can be identified early [Lik03]. As a consequence of this linkage, the generation and delivery of inventory to downstream processes can be scheduled just-in-time [Lik03, Ohn88]. A technique by which this smooth flow can be achieved is called "*Kanban*".

A Kanban system is: "*[a] production control system for just-in-time production and making full use of workers' capabilities*" [SKCU77]. Taiichi Ohno, who developed the TPS, initially conceptualized Kanban, drawing inspiration from how American supermarkets operated in the 1950s [Ohn88]: customers purchased goods in the quantities they needed, after which the "upstream" process produced what was taken. The signal passed from a downstream to an upstream process is Kanban, which means signboard or tag [Ohn88]. Since the customer initiates this process by making a request, this cascading system of signals going upstream results in a pull system. Once a task is completed, a downstream process pulls a new work item from an upstream process. On the Kanban board, the various processes are depicted as states on which the Work-

in-Progress (WIP) products pass from one state to the following [FMS14]. By restricting the number of Kanban tickets, the number of WIP products can also be limited [FMS14]. Reducing the WIP level also shortens cycle-time, which in turn helps to improve the flow of a process [Rei09]. The Kanban method typically uses a wall of cards, i.e., a Kanban board, as a visual control system [FMS14]. This form of visualization helps to identify bottlenecks as soon as they occur that need to be addressed before a new task can begin. Figure 2.5 depicts an exemplary visual representation of a Kanban board with numbers in brackets representing WIP limits.

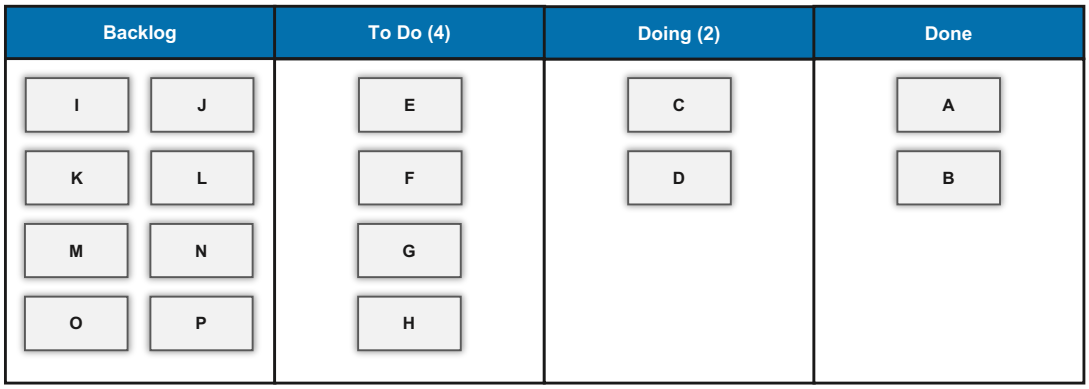


Figure 2.5.: Kanban board [Pow16]

Given the major success of the lean approach in the automotive industry, lean thinking has penetrated many sectors, including the software industry [PP03, EAO12, AMO13]. The first ideas on lean software development were developed by Mary and Tom Poppendieck [PP03] and Middleton and Sutton [MS05]. These books examined how lean thinking could be transferred from manufacturing to the more intangible world of software development [MJ11]. Lean software development, suggested by Bob Charette, draws on the success that lean manufacturing had in the automotive industry in the 1980s [CLC04]. Lean software development can be viewed as an approach applying traditional lean manufacturing [WJR07] philosophies, principles, and tools to software development [KMI15]. The term "*Lean Software Development*" has started to become a household name within the software engineering vernacular, especially in agile software development [CWR13]. In fact, when Mary and Tom Poppendieck wrote their first book on lean software development in 2003 [PP03], it was tightly connected to agile software development [EAO12]. While retaining the core intent of lean thinking, various lean principles for software development have been proposed by practitioners [WCC12]. Table 2.4 lists several lean principles that largely overlap with the core and essence of lean approaches [WCC12].

Lean and agile software development are often considered interchangeable or synonymous and appear compatible, e.g., "*ScrumBan*", a software production model based on Scrum and Kanban [Lad09], and very similar [Pet11]. For instance, both share the same people management and leadership principles and focus on quality, technical excellence, and frequent and rapid delivery of value to the customer [Pet11]. However, lean and agile software development can also be distinguished along several dimensions [Pet11, BF12]. While lean thinking focuses on the organization as a whole and aims to improve software development processes from a broader perspective [PP03, MNS12], agile methods have a single-product view, as they were initially designed for small teams [DD08]. While Lean thinking focuses on reducing costs by eliminating

Table 2.4.: Lean principles relevant to software development [PP03, Rei09, And10]

Lean software development principles [PP03]	Principles of product development flow [Rei09]	Kanban principles [And10]
• Optimize the whole • Focus on customers • Energize workers • Reduce friction • Enhance learning • Increase flow • Build quality in • Keep getting better	• Use an economic view • Manage queues • Exploit variability • Reduce batch size • Apply work-in-progress constraints • Control flow under uncertainty • Use fast feedback • Decentralize control	• Visualize the workflow • Limit work-in-progress • Manage flow • Make process policies explicit • Improve collaboratively

waste and inefficiencies, agile methods treat leanness, i.e., reducing costs by eliminating waste, as a constraint to create effective responses and valuable outcomes. Hence, lean thinking can be perceived as efficiency-oriented, while agile methods involve lean processes focusing on achieving effective results [AST06, DNBM12]. Despite these differences, there is a growing interest in finding ways to marry both worlds [DNBM12, RMO+19].

2.3. Large-Scale Agile Development

Over the past decade, agile methods have attracted considerable attention from practitioners and academics [CC19a]. It was initially believed that the benefits of agile methods were best realized in their traditional form, i.e., in small and co-located single-team projects with evolving needs [BT05, DD08, ACW09, WCP12]. Given the success of conventional agile methods at the team level, many large companies attempt to mimic this success by adopting these methods in a large-scale context [LSA11, CC19a]. However, the large-scale adoption of agile practices has proven very challenging [DD08, DPL16]. Challenges emerge due to organizational size, as difficulties in adopting agile practices increase with the growing size of the organization, i.e., products are more complex in large organizations and inter-dependencies between teams are more significant than in smaller enterprises [Amb07, DD08]. Agile methods can be introduced top-down (management-driven) or bottom-up (team-driven), making it difficult to understand the rationale for starting the change process [DPL16, CC19a]. Another challenge arises from the need for coordination and communication between multiple teams and between different organizational units that often do not operate in an agile manner, requiring additional coordination mechanisms between teams and organizational units [LMD+04]. While agile methods primarily focus on intra-team practices that work well in small projects, agile methods do not provide sufficient guidance on how agile teams should interact in large environments [Map09]. Hence, companies need to adapt the practices to their specific needs, which require additional formal communication and might compromise their agility [LMD+04]. As agile teams are often globally spread and agile methods rely primarily on frequent collaboration and communication [HC01, DPL16], applying agile practices to globally distributed projects can be difficult [HvM11].

2.3.1. Definition of Large-Scale Agile Development

While some researchers use the term "*Large-Scale Agile Development*" to describe projects with many members in a single team or projects with multiple teams over several years or a combination of size, distribution, and specialization [BBS10, DFI14], there is little agreement on what the term actually means [DM13, DFI14]. Several researchers made a first attempt and proposed several definitions of the term. Suggested definitions typically comprise the number of people or agile teams involved in the effort, the associated costs, or the project duration [DFI14]. For instance, Berger and Beynon-Davies [BBD09] categorize an agile project as large-scale whenever costs surpass 10 million GBP. Another example is provided by Bjarnason et al. [BWR11], using the project duration of more than two years as an indicator for classifying a project as large-scale. According to Paasivaara et al. [PDL08], a project with more than 40 people and seven participating agile teams can be deemed large-scale. To bring more conceptual clarity, Dingsøyr et al. [DFI14] provide a taxonomy that relies on the number of collaborating teams to define the scale of an agile project. The suggested taxonomy consists of three categories:

- *Small-scale agile projects* with one team that applies traditional agile practices, e.g., daily meetings, for intra-team coordination.
- *Large-scale agile projects* with two to nine agile teams that utilize new forums for inter-team coordination, e.g., Scrum-of-Scrums (SoS).
- *Very large-scale agile projects* with at least ten agile teams requiring several forums for inter-team coordination, e.g., multiple SoS.

Based on Dingsøyr et al. [DFI14], a project can be regarded as large-scale with at least two collaborating agile teams. Fuchs and Hess [FH18] extend this definition, noting that the term "*large-scale agile development*" can be interpreted in multiple ways: (i) the usage of agile methods in large teams, (ii) the employment of agile methods in large organizations, (iii) the adoption of agile methods in large multi-team settings, i.e., "*large agile multi-team settings*", or (iv) the application of agile practices in organizations as a whole, i.e., "*organizational agility*". Alike to Fuchs and Hess [FH18], we focus on the latter two options and use the below definition:

Definition: Large-Scale Agile Development

The term "*large-scale agile development*" is defined as the adoption of agile methods in large agile multi-team settings with at least two teams or the adoption of agile methods at the organizational level comprising multiple large agile multi-team settings.

2.3.2. Scaling Agile Frameworks

Many companies face increasing pressures and expectations to scale and there is a natural tendency to ensure team coherence by scaling agile methods to leverage the advantage of cost savings and dynamism at scale [CC19a]. Consequently, several scaling frameworks, e.g., SAFe, LeSS, and DAD, have been proposed by some custodians of existing agile methods and consultants to address issues associated with the large-scale adoption of agile practices [DMFS18, CC19a].

These frameworks incorporate several predefined workflow patterns and routines to address challenges related to large numbers of teams, inter-team coordination, and customer involvement [DPL16, CC19b]. Scaling frameworks are increasingly prevalent in contemporary large companies [LSE+13, DPL16, KHR18] as confirmed by Digital.ai's annual State of Agile survey [Dig21]. While consultants have actively promoted numerous frameworks [UKXM17, UPPM21], six of them have gained ground in practice and academia [EP17, KHR18, EWC21] (see Table 2.5).

The adoption of scaling agile frameworks not only promises benefits but has also proven to be difficult, with only a few successful cases to date [Paa17, EWC21]. This is because scaling frameworks struggle to deal with more significant complexities and inter-dependencies of large-scale, organization-wide software development efforts [EWC21]. The difficulties encountered typically center around complexities and ambiguities resulting from: (i) a large number of teams, roles, and personalities, (ii) an often unknown composition of teams and projects at the outset, (iii) abstract, epistemic, and often poorly structured work processes, (iv) multiple and often competing agendas among teams that sometimes contradict the organization itself, and (v) abstract, complex, and often unknown results and goals [RFDS16, Paa17, CC19b, EWC21].

In what follows, we will describe the three most commonly used [HPLE13] and mature [UKXM17] scaling agile frameworks, namely SAFe, LeSS, and DAD.

2.3.2.1. Scaled Agile Framework

SAFe was released in 2011 by Dean Leffingwell and was designed to bring agile methods to large enterprises. SAFe is a living framework that is continuously updated and has now reached version 5.1 [AR16, KL20]. SAFe is built on lean and agile principles and includes a collection of best practices for large enterprises [KL20]. SAFe provides companies a smooth entry into the agile world by establishing several prescriptive guidelines that are often needed to transition from a more traditional environment [KL20, RB21]. It supports companies of varying sizes, from small companies with fewer than a hundred employees to larger enterprises with thousands of employees [KHR18]. SAFe has optional extensions to enable such a degree of flexibility for large companies [KL20]. SAFe's organizational structure is large and has multiple hierarchical layers with many predefined roles and responsibilities [KHR18, KL20]. Some practitioners consider SAFe too cumbersome and complex due to its prescriptive nature, and some even say that SAFe adds complexity to bureaucracy and is becoming "*the new waterfall*" [EP17]. Four pre-built SAFe configurations allow companies to tailor SAFe to their business needs [KL20, RB21]:

- *Essential SAFe* is the most basic form of SAFe. It contains a minimal set of roles, events, and artifacts applicable at a team and program level required to transition an organization to a modern, lean way of working. It is the basic building block for all the other SAFe configurations and is the most straightforward starting point for implementation.
- *Large Solution SAFe* describes additional roles, practices, and guidance to build complex solutions. It includes Essential SAFe and extends it by a large solution level.
- *Portfolio SAFe* entails a set of competencies and practices that can fully enable business agility. It includes the components of the Essential SAFe configuration and extends it by a superior portfolio level and additional features, e.g., lean portfolio management and orga-

Table 2.5.: Comparison of major scaling agile frameworks [AR16, EP17, KHR18]

Aspect	DAD	LeSS	Nexus	SAFe	SoS	Spotify
Team / program size	200 people or more	LeSS: 2-8 teams LeSS Huge: > 8 teams	3-9 teams	50-120 people in release trains	5-10 teams	Any large project, 250 to 300 people at Spotify (30 teams)
Training / certification	Yes, multi-level certifications	Training and coaching network available	Scaled professional Scrum training & certification is available	Yes, multi-level training & certifications	None known	None known
Supported frameworks	Scrum, Kanban	Scrum	Scrum	Scrum, Kanban, Extreme Programming	Scrum	Scrum, Kanban
Required technical practices	High	Medium	Medium	High	Low	Medium
Organization type	Diverse companies	Large companies	Traditional and agile companies	Traditional and large companies	Traditional and agile companies	Only intended for Spotify; perhaps fits other agile companies
Completeness of coverage of levels	High	Medium	Medium	High	Low	Medium
Maturity level	High	High	Medium	High	Medium	Low
Complexity	Medium	Medium	Medium	High	Low	Medium
Cost of adoption	Medium	Medium	Medium	High	Low	Low
Strengths / weaknesses	☑ Lots of content ☑ Strong in areas such as architecture, design, and DevOps ☑ Incorporates many good models ☒ Vague in some areas about the “how” ☒ Can come across as a bit disjointed	☑ Non-prescriptive ☑ Gives suggestions rather clear guidelines ☑ Focus on product development based on Scrum's core ideas ☒ A “radically agile” approach that may be a hard to introduce in large traditional companies ☒ Requires perfect agile setup and experienced agile software developers	☑ Focus on integration issues ☒ Some of the parts are “secret” ☒ Scarce resources and information available	☑ “Big picture" and completeness ☑ “Softly” introducing agile at large companies ☑ Actively evolving ☑ Adaptable to different company structures ☒ Very prescriptive ☒ Not “agile enough” in its structures ☒ Top-down driven	☑ Simple, standard Scrum ☑ Focus on dependencies & resolutions ☒ Limited scaling ☒ Limited documentation ☒ Not likely “sufficient” for large-scale agile development	☑ Very agile ☑ Supports distributed teams ☑ Low adoption overhead ☒ Very limited detail about the “how” ☒ Not really a framework ☒ May only fit certain cultures

Legend:

DAD:	Disciplined Agile Delivery	Spotify:	Spotify Model
LeSS:	Large-Scale Scrum	☑:	Strength
SAFe:	Scaled Agile Framework	☒:	Weakness
SoS:	Scrum-of-Scrums		

nizational agility. The Portfolio SAFe configuration aligns portfolio execution to enterprise strategy and organizes development around one or more value streams.

- *Full SAFe* represents the most comprehensive configuration and includes all the other configurations (see Figure 2.6). It is used to maintain portfolios of large and complex solutions that typically require hundreds of people.

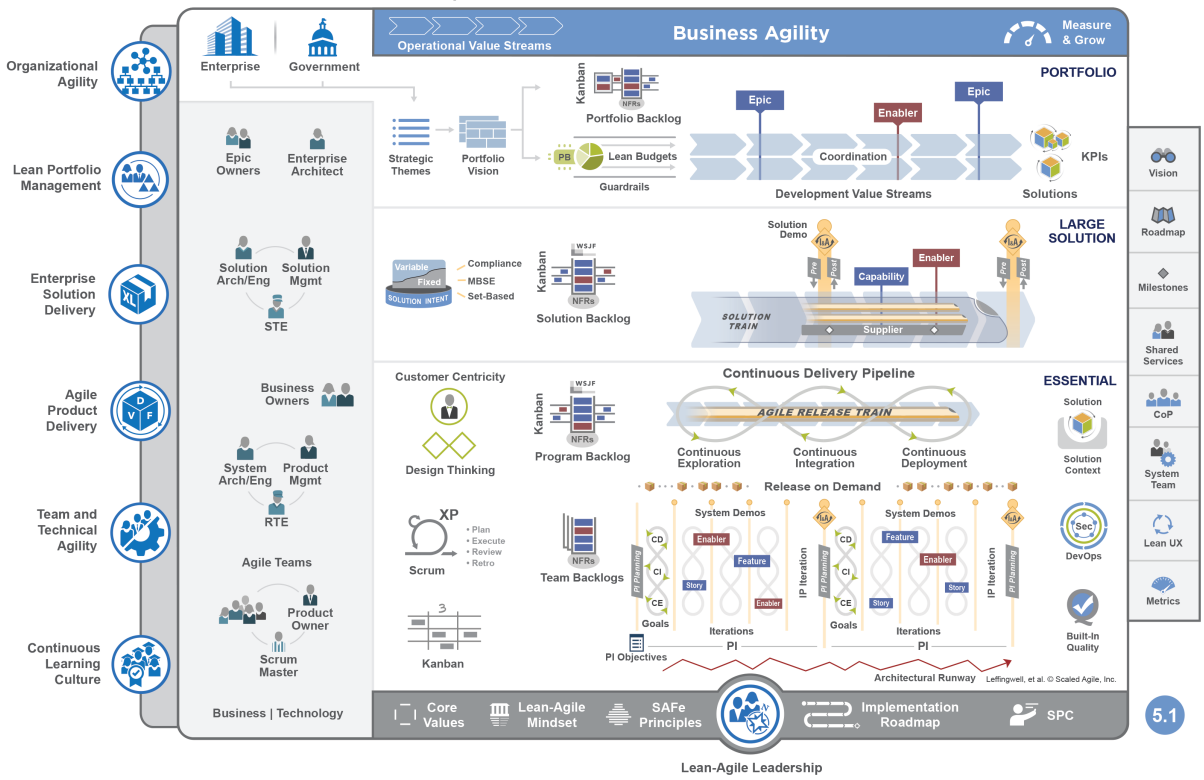


Figure 2.6.: Overview of the Full SAFe 5.1 configuration [Sca22]

Depending on the selected configuration, different layers are added. Each level integrates agile and lean practices, performs its activities, and is aligned with the other levels [AR16]. In SAFe, there are four organizational levels. The *Team Level* is the lowest level and describes how agile teams work, e.g., it suggests agile teams to use Scrum, Kanban, and XP techniques or a combination of them [KL20]. At the team level, SAFe recommends two-week iteration cycles. At the *Program Level*, all teams are part of an Agile Release Train (ART), a group of agile teams that deliver incremental releases. An ART typically consists of 5 to 12 teams or 50 to 125 individuals [PL16]. The ART develops and delivers one or more solutions in a Program Increment (PI). A PI is a time-box during which an ART produces incremental value. PIs are typically 8 to 12 weeks long [KL20]. The team and program levels are present in each SAFe configuration [RB21]. SAFe further introduces a *Large Solution Level* with additional events, roles, and artifacts. At this level, several ARTs are incorporated into a *Solution Train* that coordinates the efforts of the ARTs to deliver large and complex systems. At the *Portfolio Level*, the enterprise oversees multiple programs and guides the strategic direction of the organization

by applying lean principles. This allows executives and leaders to identify and prioritize epics and features that can be broken down and scheduled for an ART at the program level [KL20].

SAFe comprises four core components [KL20]: principles, roles, events, and artifacts (see Table 2.6), described below.

Table 2.6.: Core components of SAFe [KL20]

SAFe principles	SAFe roles	SAFe events	SAFe artifacts
• Take an economic view • Apply systems thinking • Assume variability; preserve options • Build incrementally with fast, integrated learning cycles • Base milestones on objective evaluation of working systems • Visualize and limit work-in-progress, reduce batch sizes, and manage queue lengths • Apply cadence, synchronize with cross-domain planning • Unlock the intrinsic motivation of knowledge workers • Decentralize decision-making • Organize around value	• Agile team • Scrum master • Product owner • Product manager • System architect • Release train engineer • Business owner • Solution manager • Solution architect • Solution train engineer • Epic owner • Enterprise architect	• Iteration planning • Daily stand-up • Iteration review • Iteration retrospective • Program increment planning • Agile release train sync • Scrum-of-scrums • Product owner sync • System demo • Inspect and adapt workshop • Pre- and post-program increment planning • Solution demo • Lean budget review • Communities of practice • Portfolio sync • Roadshow	• Story • Enabler story • Iteration goals • Team backlog • Feature • Enabler feature • Program increment objectives • Program board • Program backlog • Solution intent • Capability • Enabler capability • Solution backlog • Strategic themes • Portfolio vision • Portfolio epic • Lean business case • Portfolio backlog

SAFe principles. SAFe is built on ten principles that have evolved from agile and lean practices, systems thinking, and observations from successful enterprises. *Take an Economic View* promotes practices of applying an economics framework to make better decisions when building products. *Apply System Thinking* is about simplifying the solution, the enterprise building of the system, and the value streams to obtain a holistic view of solution development. *Assume Variability; Preserve Options* encourages managing the variability of agile software projects and maintaining and evaluating multiple design options and requirements during the development lifecycle. *Build Incrementally with Fast, Integrated Learning Cycles* stresses the importance of incrementally developing solutions in short iterations to allow for faster customer feedback and mitigating risk. *Base Milestones on Objective Evaluation of Working Systems* intends to provide objective milestones which evaluate the solution during the development cycle to ensure that used resources offer economic benefits. *Visualize and Limit Work-In-Progress, Reduce Batch Sizes, and Manage Queue Lengths* enables the improvement of the workflow by recommending the usage of a Kanban Board, creating smaller batches, and improving the processing rate. *Apply Cadence, Synchronize with Cross-Domain Planning* is about reducing the complexity of the development by providing a rhythmic pattern of events, adding routine and structure to the development process. *Unlock the Intrinsic Motivation of Knowledge Workers* propagates the idea of creating a better work environment to motivate workers and unleash their potential. *Decentralize Decision-Making* stresses the importance of decentralizing the decision-making process to deliver business value quickly. *Organize Around Value* encourages enterprises to organize around value to deliver and react to changing customer demands more quickly [KL20].

SAFe roles. SAFe suggests various roles depending on the scaling level. SAFe uses standard Scrum roles at the team level. An *Agile Team* is a cross-functional group of 5 to 11 people that has all the skills to define, build, test, and deliver a product in a short period. Each agile team has two specialty roles: a scrum master and a product owner. While the *Product Owner* defines stories (along with other team members) and prioritizes the team's backlog, the *Scrum Master* coaches the team, facilitates the removal of impediments, and fosters an environment for continuous improvement. SAFe proposes four roles at the program level. The *Product Manager* defines and supports the building of products. The product manager serves as the content authority for the ART and is responsible for prioritizing the program backlog. The *System Architect* defines and communicates a shared technical and architectural vision for an ART. The Release Train Engineer (RTE) is a servant-leader at the program level, coaching the ART, facilitating the ART events and processes, and assisting the teams in delivering value. The RTE communicates closely with stakeholders, escalates impediments, helps to manage risks, and fosters a culture of continuous improvement. The *Business Owner* bears the business and technical accountability for compliance, governance, and return on investment for a solution built by an ART. SAFe proposes three roles at the large solution level. The *Solution Manager* is in charge of defining and supporting the development of complex business solutions. The *Solution Architect* defines and communicates a shared technical and architectural vision within a solution train. The *Solution Train Engineer* is responsible for coaching the solution train and facilitating and guiding the work of all ARTs in the value stream. SAFe introduces two roles at the portfolio level. The *Epic Owner* defines and coordinate portfolio epics and facilitates their implementation. The *Enterprise Architect* specifies a technology strategy that enables a portfolio to support current and future business capabilities [KL20].

SAFe events. SAFe recommends various ceremonies at different levels. At the team level, SAFe suggests typical Scrum events. An *Iteration Planning* aims to organize the work and define a realistic scope for the upcoming iteration. All team members participate in this event and determine how much of the team backlog they can implement in the next iteration. A *Daily Stand-Up* is used to coordinate the team's work by answering three questions: what was done yesterday, what is being done today, and what impediments prevent completing the tasks? An *Iteration Review* is a cadence-based event where each team reviews the increment at the end of each iteration to assess its progress and adjust its backlog for the upcoming iteration. An *Iteration Retrospective* is an event where each team discusses the iteration results, reviews its practices, and identifies ways for improvement. SAFe recommends six events at the program level that primarily aim to achieve alignment between teams working on the same ART. A *PI Planning* is a two-day event where all members in the ART come together to agree on team and PI objectives. This event is used to predict which features in the program backlog will be completed and identify and manage inter-dependencies between the teams. Various roles, e.g., scrum masters and business owners, gather in an *ART Sync* and discuss the program's progress. A *SoS* is a time-boxed event that lasts between 30 to 60 minutes and usually takes place every week. During this event, the RTE and team representatives (often scrum masters) come together and coordinate the ART's dependencies and discuss the program's progress and impediments. A *Product Owner Sync* is often held by the product owners and product manager to discuss the progress toward meeting the PI objectives, discuss issues, and evaluate any scope adjustments. A *System Demo* is a bi-weekly event that provides feedback from the stakeholders

on the effectiveness and usability of the system under development. This event ensures that the integration between teams on the same ART occurs regularly. An *Inspect and Adapt Workshop* takes place at the end of the PI and is made up of three parts. First, a *PI System Demo* shows the current state of the solution and highlights the work that has been done throughout the PI. Second, during a *Quantitative and Qualitative Measurement*, the RTE presents various program metrics to the ART members. In a *Problem-Solving Workshop*, agile teams conduct a brief retrospective on the PI to identify root causes for problems and actions to address these root causes. At the large solution level, SAFe suggests two events. A *Pre- and Post-PI Planning* is used to prepare for and coordinate the PI Planning across multiple ARTs and suppliers in a solution train. This event aims to create a shared vision and mission and align on a set of features and capabilities that drive the solution. During the *Solution Demo*, the solution train's development efforts are made visible to customers and other stakeholders. At the portfolio level, SAFe recommends four events. A *Lean Budget Review* is a periodic meeting, usually taking place twice per year, to discuss how the portfolio budget should be distributed across different value streams. A *CoP* is an organized group of people who share a common interest in a particular technical or business area. A *Portfolio Sync* usually takes place monthly and provides insight into how well the portfolio progresses toward its goals. Topics discussed at this event typically include reviewing epic implementation, addressing dependencies, and removing impediments. A *Roadshow* is an enterprise-wide demonstration of work completed across several solution trains to gather feedback and input in advance of a lean budget review [KL20].

SAFe artifacts. SAFe suggests four artifacts at the team level. A *Story* is a short description of a small part of the desired functionality written in the user's language. An *Enabler Story* represents work elements that need to be done but do not directly benefit a system user, e.g., exploration, architecture, infrastructure, and compliance-related work elements. *Iteration goals* summarize business and technical goals that an agile team wants to achieve in an iteration. A *Team Backlog* contains user and enabler stories from the program backlog and stories from the team's local context. At the program level, SAFe proposes five artifacts. A *Feature* is a service fulfilling stakeholder needs. Each feature comprises a benefit hypothesis and acceptance criteria and is sized or partitioned as required to be produced by a single ART in a PI. An *Enabler Feature* is a specific work item related to exploration, architecture, infrastructure, and compliance and must be scoped to fit within a single PI. *PI Objectives* summarize the business and technical goals that an agile team or ART intends to achieve in the upcoming PI, typically created in the PI Planning. A *Program Board* is created during the PI Planning, which highlights delivery dates of the new features, dependencies of the features among the teams, and relevant milestones. A *Program Backlog* is a repository for upcoming features to address user needs for a single ART. At the large solution level, SAFe suggests four artifacts. A *Solution Intent* is a repository for storing, managing, and communicating the knowledge about the current and intended solution behavior, e.g., fixed and variable specifications and designs, references to applicable standards, and system models. A *Capability* is a higher-level solution behavior that usually spans multiple ARTs. Capabilities are split into several features to facilitate implementation in a single PI. An *Enabler Capability* is a higher-level work element that does not directly benefit the customer but completes work necessary for one or more capabilities to be subsequently implemented. A *Solution Backlog* is a repository for upcoming capabilities and enablers, each of which can span multiple ARTs. At the portfolio level, SAFe suggests six artifacts. *Strategic Themes* are busi-

ness objectives that connect a portfolio to the company's strategy. They influence the portfolio strategy and provide the business context for a portfolio's decision-making. A *Portfolio Vision* describes the future state of a portfolio's value streams and solutions. It further explains how both aim to achieve the portfolio's objectives and the broader aim of the company. A *Portfolio Epic* is a container for a significant solution development initiative and is typically cross-cutting and spanning multiple value streams and PIs. A *Lean Business Case* results from an epic analysis and is used to make a go/no-go decision for a portfolio epic. A *Portfolio Backlog* is a top-level backlog for upcoming business and enabler epics to create and evolve solutions [KL20].

2.3.2.2. Large-Scale Scrum

LeSS was published in 2008 by two agile practitioners, Craig Larman and Bas Vodde [CL13]. LeSS builds on Scrum and extends it with additional rules and guidelines for scaling agile practices without losing sight of Scrum's original goals. Larman and Vodde [CL13] describe LeSS as being "*Scrum applied to many teams working together on one product*", which is why it includes all roles, events, and artifacts recommended by Scrum. LeSS specifies additional organizational changes, e.g., cross-functional, cross-component, end-to-end feature teams, and eliminates traditional team lead and project manager roles. While SAFe takes a more prescriptive approach for completeness and guidance, LeSS takes a more lightweight and less strict approach with its "*More with LeSS*" tenet [KHR18]. LeSS specifies barely sufficient guidance needed for large-scale development and stays as agile as possible, focusing on mindset, values, and principles, while avoiding introducing too many roles, artifacts, or practices. Larman and Vodde [CL13] propose two variants of LeSS based on the size of the project. The basic LeSS framework provides guidelines and techniques for software projects that need up to eight agile teams. The LeSS Huge framework is recommended for enterprises that require more than eight teams [Vai14, KHR18]. It introduces further scaling elements needed to manage hundreds of developers, e.g., the concept of *Requirement Areas*. Requirement areas encompass major customer areas concerns from a product point of view. They may grow or shrink over time to match product needs and are organized around customer-centric requirements. All Requirement Areas follow the same cadences and strive for continuous integration across the entire project [CL13].

Figure 2.7 provides a visual representation of the basic LeSS framework. Although LeSS aims to work on principles solely, it encompasses four core components: principles, roles, events, and artifacts (see Table 2.7), described below.

LeSS principles. The ten LeSS principles provide the basis for applying LeSS in a specific organizational context. *Large-Scale Scrum is Scrum* underlines that LeSS is not a new and improved Scrum but rather is about applying the purpose, principles, and elements of Scrum in a large-scale context. *More with LeSS* indicates that LeSS is a descaling framework attempting to remove organizational complexity by solving problems in product development differently and more simply. *Systems Thinking* points out understanding and optimizing the grand scheme of the system and exploring system dynamics while avoiding local and sup-optimizations. *Lean Thinking* implies (i) creating an organizational system in which managers apply and teach systems and lean thinking and base decisions on this philosophy, and (ii) adding the two pillars of respect for people and continuous improvement toward the goal of perfection. *Empirical*

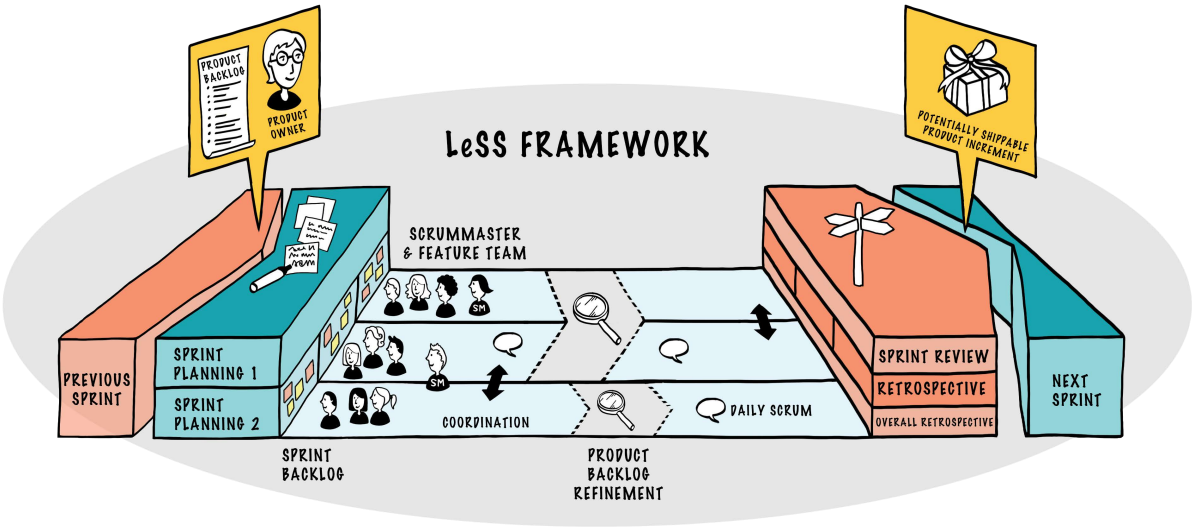


Figure 2.7.: Overview of the LeSS framework [LeS22]

Table 2.7.: Core components of LeSS [LV16]

LeSS principles	LeSS roles	LeSS events	LeSS artifacts
• Large-Scale Scrum is Scrum • More with LeSS • Systems thinking • Lean thinking • Empirical process control • Transparency • Continuous improvement towards perfection • Customer-centric • Whole product focus • Queueing theory	• Teams • (Area) product owner • Scrum master • Managers	• (Initial) product backlog refinement • Sprint • Sprint planning one • Sprint planning two • Daily scrum • Coordination and integration • Sprint review • Sprint retrospective • Overall retrospective	• (Area) product backlog • Sprint backlog • Potentially shippable product increment

Process Control is about inspecting and adapting the project, processes, organizational design, and practices based on Scrum, rather than following an exact formula. *Transparency* is about providing an unfiltered insight into project development status for all participants based on tangible done items, short cycles, and common definitions. *Continuous Improvement Towards Perfection* is about creating and delivering a product that satisfies the customer. *Customer-Centric* emphasizes the identification of value and waste in the eyes of the paying customer, reduction of the cycle-time from his/her perspective, and enhancement of feedback loops with the customer. As the customer wants the product and not parts of it, *Whole Product Focus* is about having one product backlog, one product owner, one potentially shippable product increment, and one sprint. *Queuing Theory* suggests understanding how queued systems behave and applying these insights to managing queue sizes, WIP limits, and work packages [LV16].

LeSS roles. LeSS includes three standard roles of Scrum with some additional duties and the manager's role [LV16, AR16]. A *Team* in LeSS is the same as a team in single-team Scrum. The team works closely with the customer and the product owner and coordinates and integrates

its work with other teams so that by the end of the Sprint, they have collectively produced an entire product increment. In the basic LeSS framework, a single *Product Owner* is responsible for a central product backlog and several teams. In LeSS Huge, an *Area Product Owner* focuses on one requirement area and is responsible for an area product backlog. The area product owner acts essentially the same way as the product owner in the basic LeSS framework, but with a more limited, yet still customer-centric, perspective. While the *Scrum Master* in LeSS has the same tasks as in Scrum, he/she assists one to three teams. According to LeSS, the *Manager* sees the big picture and builds the organization's capability to develop products. The manager helps the team and the scrum master remove obstacles [LV16].

LeSS events. While LeSS builds on standard Scrum events, it offers some slight modifications. An *Initial Product Backlog Refinement* is a workshop where a product's vision is defined. In this workshop, product backlog items are discovered, large items are split and refined, risks are identified, acceptance criteria are defined, and items are estimated. The initial product backlog refinement is done once a new project is started. Subsequent refinements in the upcoming Sprints are made in regular product backlog refinement workshops. A *Sprint* in LeSS is no different from a single-team Scrum and is a container for all other events. In LeSS, there is a change in the sprint planning meeting, divided into two parts. *Sprint Planning One* is a meeting for all teams to decide which team works on which part of the product backlog. Two members and the product owner represent each team to determine what part of the product backlog items to work on [LV16, AR16]. *Sprint Planning Two* is essentially the same as in single-team Scrum, where each team does its sprint planning and creates its plan for completing its tasks during the sprint. A *Daily Scrum* is held per team where it spends 15 minutes and each member answers three questions: what was done yesterday, what will is being done today, and what is in progress [LV16]. A *Coordination and Integration* between teams aims to improve the information sharing and collaboration, which can be done regularly during the Sprint in various formats, e.g., open spaces, town halls, or SoS [LV16, AR16]. A *Sprint Review* is a single occasion for all teams to review the product increment. At the end of the sprint, all teams have their individual *Sprint Retrospective*. During each sprint retrospective, the teams brainstorm about obstacles impeding them and all the other teams. An *Overall Retrospective* is a new event in LeSS whose purpose is to discuss the project's cross-team, organizational, and systemic issues [LV16].

LeSS artifacts. In LeSS, except for a small extension of the product backlog in LeSS Huge, there are no differences to Scrum. In the basic LeSS framework, multiple teams work on a single *Product Backlog* defining all work items on the product. In LeSS Huge, an *Area Product Backlog* is created for each requirement area, which the area product owner manages. An area product backlog is a view into the product backlog based on the requirement area. A *Sprint Backlog* is a list of work that the team needs to do to complete the selected product backlog items. The sprint backlog is per team. The result of each sprint is a *Potentially Shippable Product Increment*. The work of all teams must be integrated before the end of each sprint [LV16].

2.3.2.3. Disciplined Agile Delivery

In 2009, Scott Ambler and Mark Lines began developing DAD, officially introduced in 2012 with the book "*Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery*

in the Enterprise" [AL12]. DAD is a hybrid process that extends Scrum to cover the entire delivery life-cycle while adopting further proven agile and lean practices, e.g., XP and Kanban [AL12, AR16]. Comparing DAD to other scaling frameworks shows that it gives teams the freedom to customize processes depending on their needs. DAD is handy for having guidance related to architecture and design [AR16].

DAD entails four core components [AL12]: principles, phases, life-cycles, and roles (see Table 2.8), described in the following.

Table 2.8.: Core components of DAD [AL12]

DAD principles	DAD phases	DAD life-cycles	DAD roles
• Delight customers • Be awesome • Be pragmatic • Context matters • Optimize flow • Choice is good • Enterprise awareness	• Inception phase • Construction phase • Transition phase	• Agile life-cycle • Lean life-cycle • Continuous delivery: agile life-cycle • Continuous delivery: lean life-cycle • Exploratory (lean startup) life-cycle • Program life-cycle	• Stakeholder • Product owner • Team member • Team lead • Architecture owner

DAD principles. The seven principles of DAD provide a philosophical foundation for business agility based on lean and flow concepts. *Delight Customers* aspires to go beyond satisfying customer needs and meeting their expectations. *Be Awesome* is about striving to be the best and always getting better. *Be Pragmatic* states that the main goal is not just to be agile but to achieve continuous improvement by applying agile, lean, or traditional strategies when they make the most sense in a given context. *Context Matter* is about choosing the way of working that reflects the given context and evolves as the context evolves. *Optimize Flow* targets optimizing the flow across the value stream to meet customer needs. *Choice is Good* states that different techniques should be considered and chosen according to the situation, rather than obsessively sticking to practices that are not helpful in a given context. *Enterprise Awareness* appeals to looking beyond local needs and considering the organization's long-term needs [AL20].

DAD phases. DAD covers three project life-cycle phases. In the *Inception Phase*, the team performs lightweight visioning activities, e.g., defining the scope and goal of the project. In the *Construction Phase*, the team develops a product that provides business value and addresses any issues in the early development phase of the product by improving quality and testing the architecture early. In the *Transition Phase*, the team makes arrangements to get the product into the hands of the customer [AL12, AR16].

DAD life-cycles. DAD offers six different life-cycles based on a company's needs as its basic assumption is that one process size does not fit all. The *Agile Life-Cycle* builds on Scrum and XP. Typical scenarios for applying this life-cycle include situations related to improvements or new features or where work can be identified, prioritized, and estimated early. The *Lean Life-Cycle* promotes lean principles, e.g., maximizing flow and reducing bottlenecks. In this life-cycle, a team pulls new work from the work item pool when it has capacity. While Scrum mandates the usage of a series of events, e.g., sprint planning, daily scrum, and sprint review, lean principles do not prescribe this and suggest that they only be done as needed. The lean

life-cycle is appropriate when work can be broken down into smaller items of roughly the same size or is difficult to predict in advance. The main difference between this and the agile life-cycle is that new features are released at the end of each sprint rather than after several sprints. In this life-cycle, teams need a mature stage of continuous integration and delivery practices. The *Continuous Delivery: Lean Life-Cycle* enables the goal of delivering increments of the solution more frequently than the other life-cycles. It demands a mature stage of technical infrastructure and continuous integration and deployment practices. This life-cycle is best suited when solutions can be delivered frequently and incrementally or when the team is long-lived and works on several releases over time. The *Exploratory (Lean Startup) Life-Cycle* builds on lean startup principles and extends them with ideas from complexity theory to increase its effectiveness. The philosophy of this life-cycle is to minimize up-front investments in solutions and instead conduct small experiments. The exploratory (lean startup) life-cycle is useful when the solution targets a very uncertain case, e.g., a new, unexplored market or a new product, or when the team is willing to experiment and evolve its idea. The *Program Life-Cycle* organizes the workflow of multiple teams and provides an organizational structure to support their coordination within the program. In this life-cycle, teams are free to choose their ways of working. Similar to SAFe and LeSS, the Program Life-Cycle can be applied to support many agile teams working together. The Program Life-Cycle is helpful when developing complex solutions that require work of multiple teams or when the solution is a platform or collection of related solutions [AL12, AL18].

DAD roles. DAD includes five primary roles, three of which are similar to roles in Scrum. DAD adds the role of the *Stakeholder*, who is significantly affected by the outcome of the solution. The stakeholder is more than an end-user, e.g., a senior manager, an operations staff member, a person funding the project, an auditor, or a program manager. The team ideally works with its stakeholders on a daily basis throughout the project. The *Team Member* focuses on creating the solution for the stakeholders and performs various construction activities, e.g., analysis, architecture, design, programming, testing. The team member is sometimes referred to as the developer or the programmer in core agile practices. However, DAD recognizes that not every team member necessarily writes code. The *Product Owner* represents the voice of the customer speaking to the team. The product owner advocates the needs of the stakeholder community. The product owner clarifies all details related to the solution and maintains a prioritized list of work items that the team implements. Each team, or team of teams in the case of large programs, has a single product owner. The product owner is responsible for organizing demonstrations of the evolving solution and communicating the project's status to key stakeholders. The *Team Lead* is comparable to a scrum master and supports the team in executing technical management activities. The team lead is a servant-leader and creates and maintains the conditions that enable the team to succeed. The team lead is also an agile coach, helping the team focus on delivering work items and meeting sprint goals and commitments made to the product owner. The team lead facilitates communication, empowers the team to self-optimize its processes, ensures that the team has the resources it needs, and removes any obstacles. The *Architecture Owner* is responsible for the team's architecture decisions and supports the creation and evolution of the overall solution design. In smaller agile teams, the person in the team lead role often takes over the role of the architecture owner. Although the architecture owner is usually the senior developer, it is not a hierarchical position. In addition to the five primary roles, DAD proposes

five supporting roles that are temporarily introduced to address scaling issues, i.e., the role of the *Specialist*, *Domain Expert*, *Technical Expert*, *Independent Tester*, and *Integrator* [AL18].

Due to the similarity of the program life-cycle to other scaling frameworks, we briefly describe it below. A visual representation of it is shown in Figure 2.8.

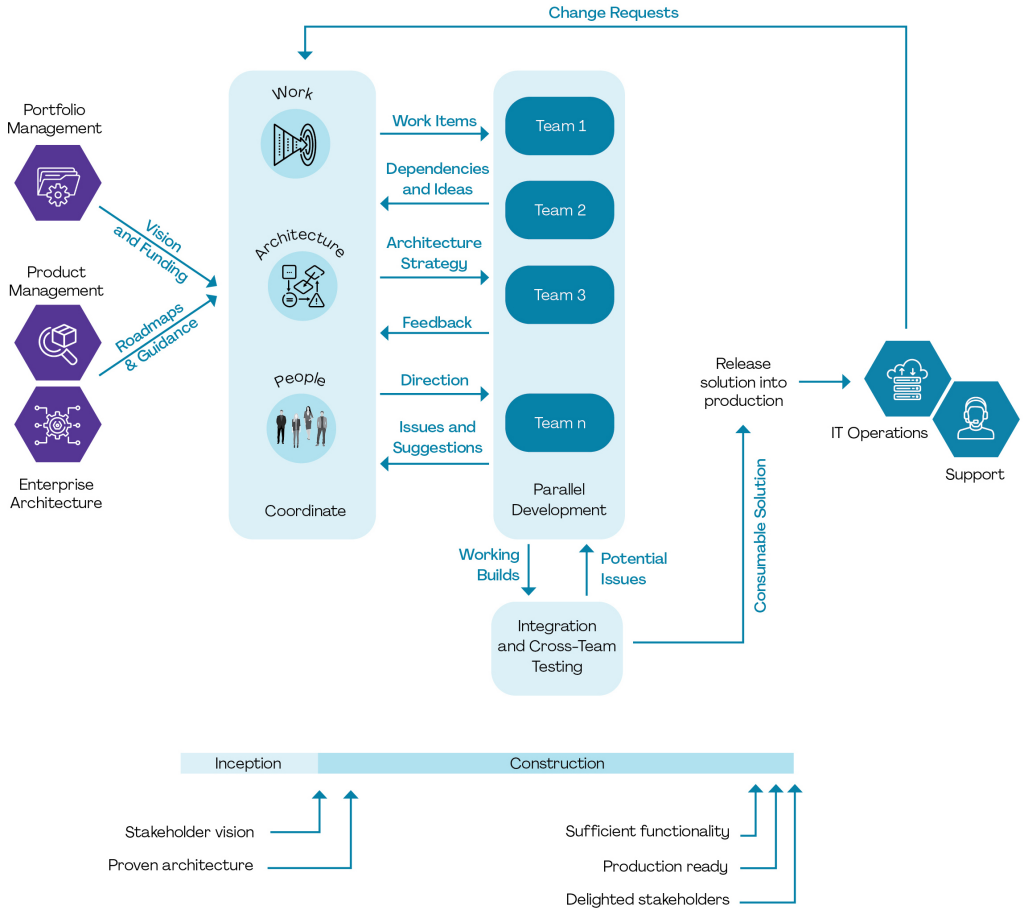


Figure 2.8.: Overview of DAD's program life-cycle [Pro22]

There is an explicit inception phase in the program life-cycle, as some up-front time needs to be invested in getting organized, especially for large teams. In this life-cycle, teams choose and evolve their ways of working, including selecting their life-cycles and practices. The teams may have some arrangements to align within the program, e.g., following common guidelines and strategies. A team can be either a feature team or a component team. While a feature team works on vertical functionality pieces, implements user stories, or handles change requests, a component team works on a specific aspect of the system, e.g., security features, transaction processing, or logging. The teams have to coordinate at three levels: coordinating the work, technical/architectural issues, and people issues. This coordination is done by product owners, architecture owners, and team leads. Product owners address work/requirements issues to ensure that each team is doing the right work at the right time. Similarly, architecture owners evolve the architecture over time, and team leads resolve cross-team issues. In the program life-cycle,

system integration and testing occur in parallel. This life-cycle envisages a separate team to perform system integration and cross-team testing [AL12, AL18].

2.4. Patterns

The documentation of best practices to recurring problems in a specific context by so-called "*patterns*" is a widely accepted way of facilitating knowledge abstraction and dissemination in design-intensive domains [Buc11]. The primordial idea of using patterns originated from the architecture discipline and was initially introduced by Alexander et al. [Ale64, AIS77], coining the following definition of a pattern [AIS77]:

Definition: Pattern

Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.

Alexander et al. [Ale64, AIS77] laid the foundation for the use of patterns in computer science. Since then, related fields have successfully adopted the idea of patterns, e.g., software engineering by Gamma et al. [GHJV94], software architectures by Buschmann et al. [BMR+96], or project management by DeMarco et al. [DHL+08]. In all these disciplines, patterns describe operational knowledge gained from practice, which are neither invented nor create new knowledge but rather represent observations [Kel12]. Instead, patterns describe solutions from practice that are not captured in full detail as they are observed but in an appropriate abstraction to make the solutions accessible to a broader range of people [Cop96, Kel12]. Coplien [Cop96] posits the so-called "*rule of three*", which states that an observed solution must refer to at least three known uses to be called a pattern. Before that, an observed solution represents a pattern candidate.

Various ways of pattern documentation, known as pattern forms, have been proposed. Popular pattern forms include the Alexandrian Form, Gang of Four Form, and Coplien Form, to name a few [Fow06, Ern10]. All pattern forms have specific advantages and limitations depending on the context in which they are applied [Ern10]. Since there is no ideal pattern form, authors must consider their experiences, intentions, and target audience when choosing an existing form or creating a new one [BHD07, Ern10]. According to Fowler [Fow06], this choice is a personal one and should also consider one's writing style and the ideas to be conveyed. Although various pattern forms exist, five essential elements are present in all of them. Buschmann et al. [BHD07] provide an overview of these elements and their purpose:

- A *pattern name* identifying the pattern and making it memorable, usable, and distinct.
- A *context description* representing the surrounding situations leading to a problem to which the pattern applies.
- A *problem description* specifying the problem the pattern should address.
- A *solution description* entailing the elements of the solution design, their responsibilities, relationships, and collaborations.

- A *consequences description* representing the results and trade-offs of applying the pattern, including its advantages and disadvantages.

Usually, patterns are not developed individually but as part of a pattern language [CH04], as applying single patterns is insufficient to build complex real-world systems or organizations [DSRB00]. In situations where a solution to a single problem is too complex to be documented by a single pattern, or the resulting pattern would be too intricate, a pattern language can be used as an alternative. A pattern language breaks down the complex problem/solution description into multiple self-contained patterns [MD97]. Pattern languages connect different patterns and help understand the collection of patterns as a whole [AIS77, Kel12]. As each pattern solves a specific problem within a pattern language, references between the independent patterns are necessary. Such references can be used to identify a smaller pattern used by a larger pattern, define variants of patterns, or define a sequence of elaborations [Nob98].

2.5. Communication Networks

Communication can be understood as a means of transmitting information, e.g., thoughts, ideas, and emotions, between sender(s) and receiver(s) [GS05]. The communication flows connecting senders and receivers are called communication networks [Lun11]. To understand communications networks, researchers often analyze the structure of these networks and the locations of nodes in these networks [AAH11]. Figure 2.9 shows five commonly observed communication network structures that differ in the degree to which they are centralized or decentralized [Ram11]. These communication network structures are explained below [Lun11]:

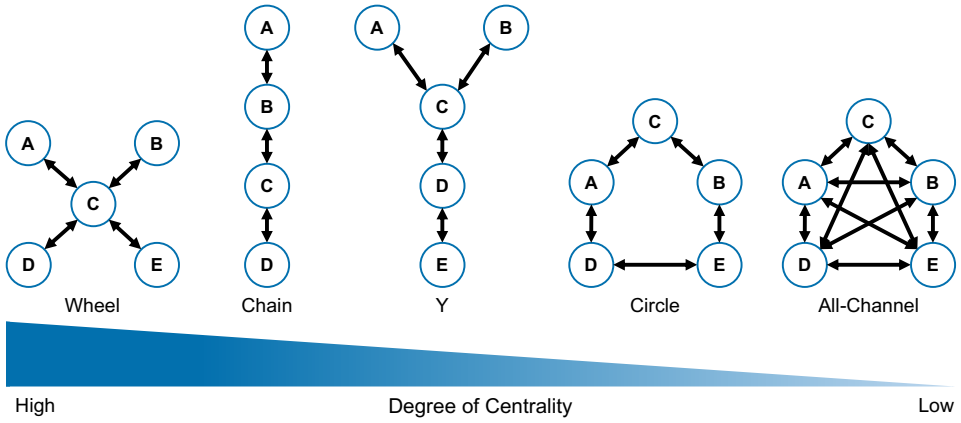


Figure 2.9.: Common communication network structures [Lun11]

- The *wheel network* represents a two-level hierarchy and is the most centralized and structured communication network pattern, as each member can only communicate with one other member. The superintendent C receives all information from his subordinates A, B, D, and E and sends information back, usually in the form of decisions.
- The *chain network* is the second-highest centralized communication network pattern. In

this network, only two members communicate with each other, and they, in turn, have only one member with whom they communicate. Information is usually transmitted in such a network in the form of relays.

- The *Y network* is comparable to the chain network pattern, except two members are not in the chain. Members A and B can send information to C in the Y network, but they cannot receive information from anyone else. Members C and D can exchange information. Member E can exchange information with member D.
- The *circle network* represents a three-level hierarchy and stands for horizontal and decentralized communication that provides equal communication opportunities for each member. Each member can communicate with the members to their right and left in this network. The members have the same constraints, but the circle network is less constrained than the wheel, chain, or Y networks.
- The *all-channel network* represents an extension of the circle network in that it connects all members of the circle network. The all-channel network allows members to communicate freely with all other members (decentralized communication). The all-channel network has no central position, and there are no communication restrictions for any member.

Another method besides analyzing the structure of communication networks is to evaluate the location of nodes in these networks. Measuring the network location is about determining the centrality of a node that, in turn, helps to assess the importance of a node in the network [Fre78, AAH11]. A node can be central from a local or global perspective. A node is locally central if it has a sizeable direct neighborhood of nodes [AAH11]. If a node is globally central, it has a position of strategic significance in the overall structure of the network [Sco91]. A way of measuring node centrality is to measure the degree of the node in the graph. A node's degree is the number of other nodes directly connected to that node [AAH11]. Since the degree of a node is calculated based on the number of its neighboring nodes, the degree can be viewed as a measure of local centrality [Sco91]. The degree centrality of node p_k is given by [AAH11]:

$$C_D(p_k) = \sum_{i=1}^{n} a(p_i, p_k) \tag{2.1}$$

where n is the number of nodes in the network and $a(p_i, p_k)$ is a distance function. $a(p_i, p_k) = 1$, if node p_i and node p_k are connected, otherwise $a(p_i, p_k) = 0$ [AAH11].

A drawback of the regular degree centrality measure is that it only allows the comparison of nodes in networks of the same size [AAH11]. To have a more general measure for comparing the degree centrality of nodes of different networks with different sizes, Freeman [Fre78] suggests a relative measure. This measure normalizes the actual number of links by the maximum number of connections it can have. The normalized degree centrality of node p_k is given by [AAH11]:

$$C'_D(p_k) = \frac{\sum_{i=1}^{n} a(p_i, p_k)}{n-1} \tag{2.2}$$

CHAPTER 3

Research Design

The research design of a research endeavor represents a blueprint for research activities to be undertaken to fulfill the research objectives and answer the research questions satisfactorily [Bha12]. Two crucial aspects of the research design include selecting a research strategy (see Section 3.1) and research methods (see Section 3.2) properly [Bha12].

3.1. Research Strategy

Depending on a researcher's interest, a scientific inquiry can take two possible forms. First, in inductive research, the goal is to derive theoretical concepts and patterns from observed data. Second, in deductive research, the goal is to test concepts and patterns known from theory using new empirical data [Bha12]. Following the goals above, an adequate strategy of inquiry, i.e., research method, has to be selected that moves from the underlying philosophical assumptions to the research design and collection of data [Mye97]. The choice of the inquiry strategy highly affects the way the researcher collects data. Three different inquiry strategies exist in behavioral research: quantitative strategies, qualitative, and mixed strategies [CC18].

- *Quantitative strategies* strive to understand and interpret quantitative data and test theories by examining the relationship between variables [Yıl13, CC18]. These variables can be measured, usually with instruments, so that the data can be analyzed using statistical techniques. Surveys and experiments are examples of quantitative methods [CC18].
- *Qualitative strategies* aim to comprehend and explain complex social and organizational phenomena. Due to the phenomena's complexity, qualitative research is often limited to specific units of analysis that do not aim at generalizing results [SC90]. Case studies and expert interviews are famous examples of qualitative methods [Bha12].

- *Mixed strategies* target to combine the strengths of quantitative and qualitative strategies by collecting both quantitative and qualitative data, integrating the two types of data, and applying different designs that may incorporate philosophical assumptions and theoretical frameworks [JOT07, CC18]. The basic assumption of this form of inquiry is that integrating qualitative and quantitative data yields further insights beyond the information provided by either the quantitative or qualitative data alone [CC18].

This dissertation follows an inductive research strategy [Bha12] using a five-fold approach combining evidence-based research (structured literature review and systematic mapping study), qualitative research (case study), quantitative research (survey), and design science research (PDR research). Through this mixed-methods approach, we intend to derive unique and rich insights [VBB13] regarding the large-scale adoption of agile methods. First, we applied an evidence-based approach, namely a systematic mapping study, to explore the state of the art in large-scale agile development and reveal research gaps (see P1). Second, we first conducted a structured literature review to compile a list of existing scaling frameworks (see P2) and surveyed their inventors to examine their raison d'être and the benefits and challenges of their adoption (see P3). We then conducted a single-case study to gain additional insights into the real-world adoption of a well-known scaling framework (see P4). Third, we performed a structured literature review to create a list of typical challenges stakeholders face in large-scale agile development (see P5) and then used the PDR method to propose solution artifacts to address these reported challenges (see P6 and P7). Fourth, we conducted a single-case study (see P8) and a multiple-case study (see P9 and P10) to explore the collaboration between architects and agile teams in large-scale agile development endeavors.

3.2. Research Methods

This section elaborates on the five used research methods: systematic mapping study (see Section 3.2.1), structured literature review (see Section 3.2.2), survey research (see Section 3.2.3), case study research (see Section 3.2.4), and PDR research (see Section 3.2.5). Thereby, we briefly introduce these methods by describing their general information and characteristics, the necessary steps for conducting them, and how their usage contributed to the results of this dissertation. The application of these methods is described in detail in each publication in Part B of the dissertation. Table 3.1 shows the mapping between the embedded publications in the dissertation and the underlying research methods.

3.2.1. Systematic Mapping Study

Inspired by the evidence-based paradigm in medical research [KDJ04], [Kit04] adapted the concept of evidence-based research to the field of software engineering. According to Kitchenham et al. [KDJ04], evidence-based software engineering aims "*to provide the means by which current best evidence from research can be integrated with practical experience and human values in the decision-making process regarding the development and maintenance of software.*" The systematic literature review is the primary instrument for achieving this goal [KBB15]. The systematic

Table 3.1.: Overview of research methods applied in the embedded publications

RQ	No.	Title	PRS	SES	MCS	PDR	SCS	SLR	SMS	SUR
RQ1	P1	Revealing the State of the Art of Large-Scale Agile Development Research: A Systematic Mapping Study		●					●	
RQ2	P2	Investigating the Role of Architects in Scaling Agile Frameworks		●				●		
	P3	Evolution of the Agile Scaling Frameworks	●							●
	P4	Investigating the Adoption and Application of Large-Scale Scrum at a German Automobile Manufacturer	●				●			
RQ3	P5	Identifying and Structuring Challenges in Large-Scale Agile Development Programs based on a Structured Literature Review		●				●		
	P6	Documenting Recurring Concerns and Patterns in Large-Scale Agile Development	●			●				
	P7	Identifying and Documenting Recurring Concerns and Best Practices of Agile Coaches and Scrum Masters in Large-Scale Agile Development	●			●				
RQ4	P8	Using Social Network Analysis to Investigate the Collaboration Between Architects and Agile Teams: A Case Study of a Large-Scale Agile Development Program in a German Consumer Electronics Company	●				●			
	P9	What to Expect from Enterprise Architects in Large-Scale Agile Development? A Multiple-Case Study	●		●					
	P10	Investigating the Role of Enterprise Architects in Supporting Large-Scale Agile Transformations: A Multiple-Case Study	●		●					

Legend:

PRS:	Primary study	SCS:	Single-case study
SES:	Secondary study	SLR:	Structured literature review
MCS:	Multiple-case study	SMS:	Systematic mapping study
PDR:	Pattern-based design research	SUR:	Survey

mapping study is relatively new in software engineering and can be considered an alternative [KBB15]. The systematic mapping study represents a second tool for evidence-based software engineering and a secondary study that aims to obtain a comprehensive overview of a specific research topic, identify research gaps, and gather evidence to guide future research endeavors [KC07, KBB15]. Systematic mapping studies are similar to systematic literature reviews in that they use many methodological elements of systematic literature reviews. There are also some notable differences between those two methods. First, systematic mapping studies are conducted at a higher level of granularity and can deal with research areas that are broad and poorly defined [KBB15]. Hence, systematic mapping studies differ from systematic literature reviews in that they are less complex, examining broader research questions that do not require in-depth but high-level analysis [KBB10]. Second, systematic literature reviews [KC07, BKB+07] are typically applied to identify, evaluate, and compare all available research on a particular research question. However, systematic mapping studies aim to map the research undertaken rather than answering a detailed research question [BKB+07]. A well-established five-step approach for conducting systematic mapping studies in the software engineering context is proposed by Petersen et al. [PFMM08] (see Figure 3.1), detailed below.

Scope definition phase. The first phase of the systematic mapping study begins with the

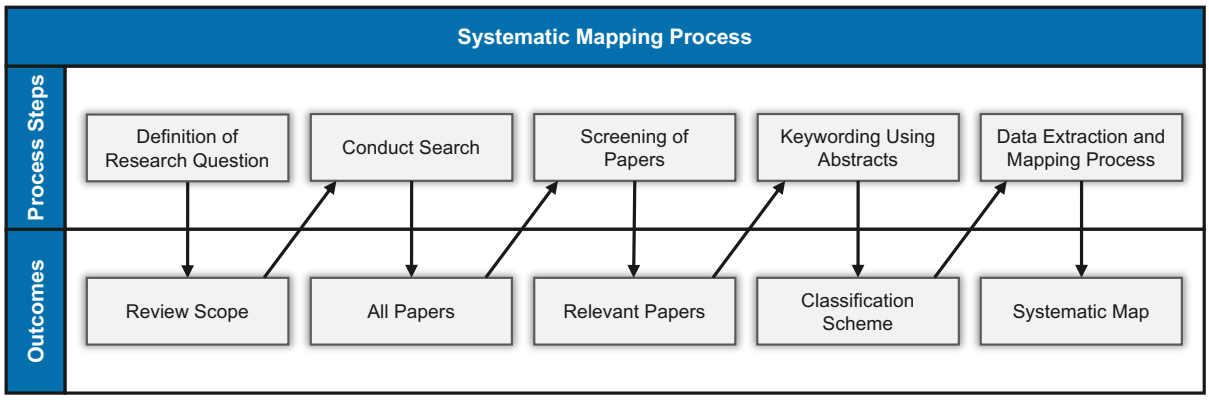


Figure 3.1.: Systematic mapping approach [PFMM08]

formulation of the systematic mapping study's research objectives, e.g., providing an overview of a research area or mapping the frequencies of publications over time to identify trends, which are then reflected in the defined research questions [PFMM08].

Literature search phase. The literature search phase is an iterative process. It is dedicated to defining a relevant search string based on the elicited research questions and automatically searching for relevant studies using search strings in scientific databases or manually searching relevant conference proceedings or journals [PFMM08]. A suitable method for creating the search strings is to structure them in terms of population, intervention, comparison, and outcome [KC07]. Of course, the formulated research questions should drive their structure [PFMM08]. In the next step, the most complete scientific databases and indexing systems should be selected and exercised to collect all publications relevant to the study. Factors such as accessibility, ability to export search results, indexing of peer-reviewed publications, and reputation can be used to select adequate scientific databases and indexing systems [BKB+07]. Relevant conference proceedings and journals should also be selected and manually searched for related studies [PFMM08].

Paper screening phase. In the paper screening phase, the retrieved papers are first filtered by removing possible duplicates and then screened based on their relevancy using inclusion and exclusion criteria to exclude studies not relevant to answering the research questions [PFMM08].

Abstract keywording phase. This phase is about classifying relevant papers by reading their abstracts and performing keywording. Keywording is a way to reduce the time required to develop a classification scheme and ensure that the scheme considers existing studies. The keywording process involves two steps. The first step comprises reading the abstracts and searching for keywords and concepts that reflect the contributions of the studies. In the second step, keywords from the various papers are combined to develop a high-level understanding of the nature and contribution of the research. Once a final set of keywords is selected, they can be clustered and used to form the categories for the systematic maps [PFMM08].

Data extraction and mapping phase. In this phase, the information and relations contained in the search results are analyzed to extract relevant information. The data extraction process involves classifying the relevant articles into a classification scheme that may evolve during the

process. The extracted data can be presented in systematic maps that visualize the results with graphs, tables, or other graphical representations [PFMM08].

In P1, we use a systematic mapping study approach to analyze existing research on large-scale agile development, provide an overview of the state of the art, and reveal future research areas.

3.2.2. Structured Literature Review

Literature reviews are vital and appropriate means for scientists to summarize and progress the current state of knowledge in a structured and replicable way on a specific area of interest [Coo88, Bak00, WW02, RS04]. A literature review is more than simply searching for relevant studies and collecting summaries of many papers [LE06]. Instead, a literature review strives to assess, structure, and reveal the most relevant scientific publications within a specific domain [TDS03]. According to Webster and Watson [WW02], good literature reviews create a solid foundation for advancing knowledge and facilitating theory development. They also close areas where an abundance of research exists and uncover areas where further research is needed. Consequently, the process of identifying sources for relevant and high-quality literature, searching for relevant literature, and then analyzing and synthesizing the findings from the selected literature must be made as transparent as possible by researchers to demonstrate credibility [VBSN+09]. Hence, reliable literature reviews follow a systematic process to increase rigor, validity, and relevance [TDS03, VBSN+09]. Although several articles have been published that provide guidelines for conducting systematic literature reviews (cf. [WW02, KC07]), we opted for the structured literature research approach proposed by Brocke et al. [VBSN+09] in two publications, P2 and P5, since its recipe-alike structure covers all the essential stages of a literature review. This approach comprises five phases [VBSN+09] (see Figure 3.2), which are further detailed below.

Scope definition phase. An essential part of any literature review is defining its scope and purpose [VBSN+09]. For this purpose, a taxonomy proposed by Cooper [Coo88] can be used, which comprises six constituent characteristics:

- *Focus* is the key area of interest to the reviewer. This area can involve the following: research outcomes, research methods, theories, practices, or applications.
- *Goal* refers to the reviewer's expectations that the review should fulfill.
- *Organization* concerns the way the reviewer organizes his or her search study. The literature review may be organized chronologically, conceptually, or methodologically.
- *Perspective* is the reviewer's point of view in discussing the literature. The reviewer might introduce the study with a neutral position or an espousal position.
- *Audience* refers to the groups of people to whom the review is directed.
- *Coverage* regards how the reviewer searches the literature and how he or she makes decisions about the appropriateness and quality of documents.

Topic conceptualization phase. According to Torraco [Tor05], a literature review should start by getting a broad idea of what is known about the topic and what areas may still need knowledge. Therefore, the topic conceptualization phase aims to narrow down potential expres-

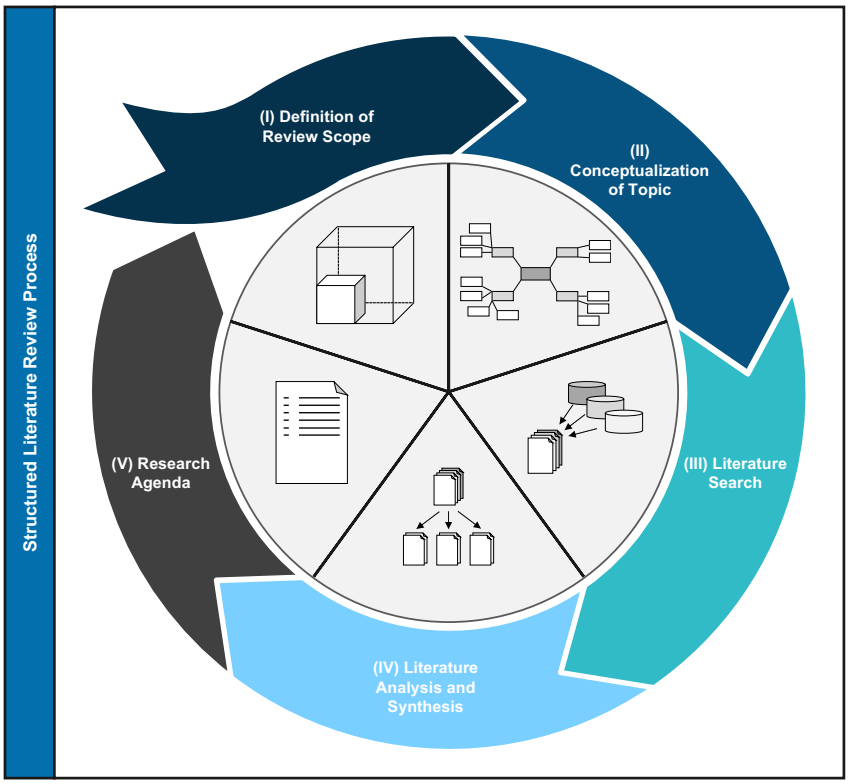


Figure 3.2.: Structured literature review approach [VBSN+09]

sions, terms, and search phrases to the most relevant key terms, representing the key deliverables of this phase. Working definitions [ZC06], seminal textbooks, encyclopedias, handbooks [Bak00], and concept mapping can be used to identify key terms [VBSN+09]. Concept mapping also provides the opportunity to uncover relevant search terms, mainly related concepts or synonyms and homonyms, that can be used in the subsequent literature search [VBSN+09].

Literature search phase. This phase includes database, keyword, backward, and forward searches, and an ongoing evaluation of sources [VBSN+09]. The following four steps are performed in the literature search process [VBSN+09]: (i) accumulating related research outlets that may cover relevant papers, (ii) selecting adequate databases containing these outlets, (iii) conducting keyword searches by modifying the identified search algorithms in each of the selected databases, and (iv) executing a forward and backward search based on the derived articles.

Literature analysis and synthesis phase. After collecting a sufficient number of papers, the literature analysis and synthesis phase embraces the systematic analysis of the identified papers [VBSN+09]. The identified articles are first reviewed based on their titles, keywords, and abstracts as part of the assessment. The final bulk of relevant and promising articles are then examined using full-text analysis. Finally, key findings, commonalities, and insights are derived from the analysis during this phase [VBSN+09].

Research agenda phase. In the last phase, a research agenda is created by collecting and

summarizing future research questions descended from the found papers and conclusions drawn by the researcher [WW02, VBSN+09].

In P2, we use a structured literature review approach to compile a list of existing scaling agile frameworks and to assess their maturity based on information deduced from the literature. In P5, we again employ a structured literature review approach to reveal typical challenges of stakeholders in large-scale agile development reported in the literature.

3.2.3. Survey Research

Researchers conduct and validate their solutions using empirical research as the need for empirical investigations in software engineering increases [GPRN18]. Survey research is one of the empirical research methods. It aims to obtain information by gathering data from a specific sample of a large population through personal or impersonal means to study its characteristics [IM95, Kas05]. Since researchers encounter issues when conducting surveys in software engineering [GPRN18], Linåker et al. [LSMdMH15] propose an approach consisting of eight steps (see Figure 3.3), depicted below.

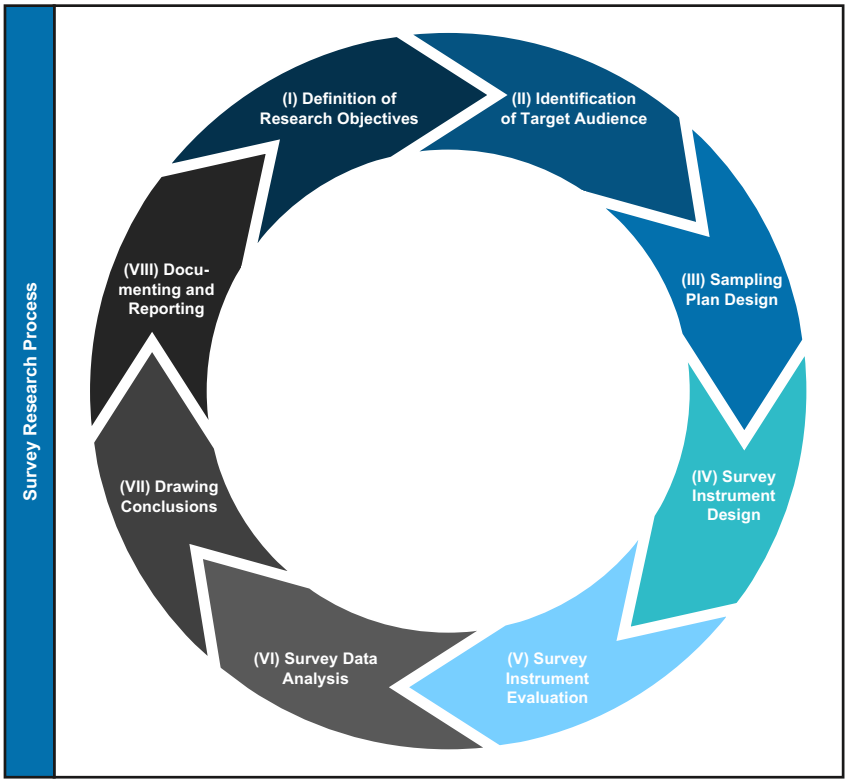


Figure 3.3.: Survey research approach [LSMdMH15]

Scope definition phase. The scope definition phase is about defining the research objectives, which describe the issue of interest and assist the establishment of the research scope and context for formulating the research questions [LSMdMH15, GPRN18]. As part of the research

objectives, not only should the research questions be defined, but they also should: (i) explain the motivation behind the survey [CLVB03], (ii) consider the required resources to achieve the objectives [KP08], (iii) consider the targeted respondent population of the survey [CLVB03, Kas05], (iv) discuss the way the data obtained from the survey should be used [Kas05], and (v) consider possible areas close to the research objectives that were left unstudied [KP08]. Related work should also be considered when defining the survey's research objectives [CLVB03, Kas05]. The aim of a survey can be either exploratory, descriptive, or explanatory [WRH+12]:

- *Exploratory surveys* support researchers in breaking new ground and discovering new insights into an area that is unknown to some degree.
- *Descriptive surveys* help researchers to make statements or assertions about a particular population.
- *Explanatory surveys* assist researchers om explaining trends, phenomena, or problems observed in the population.

Target audience identification phase. This phase is concerned with defining the target population and audience [LSMdMH15]. According to Linåker et al. [LSMdMH15], the selection of the target audience must be guided by the research objectives. In this sense, the survey instrument must be designed from the respondents' perspective [LSMdMH15]. In addition, the choice of survey method, e.g., interview or web questionnaire, must be chosen, taking into account which of them might be more accessible to the target audience [LSMdMH15]. The target audience in a software engineering survey can be characterized by some primary attributes, usually related to the respondents' background or including demographic attributes, e.g., organizational size, responsibilities, gender, age, and relevant experience [Kas05, LSMdMH15].

Sampling plan design phase. This phase is about selecting a sample to study the characteristics of the population. Sampling is needed to characterize a large population when selecting all units from a sampling frame is not feasible [HATB95]. Sampling can be mainly divided into two types [KPP+02], namely non-probabilistic and probabilistic sampling. Non-probabilistic sampling refers to all sampling approaches in which randomness cannot be observed in selecting the units, i.e., the units from a sampling frame do not have the same probability of being selected [Kas05, FJ13, Tho14]. There are four non-probabilistic sampling techniques: convenience sampling, judgment sampling, quota sampling, and snowball sampling. All units in a sampling frame must have the same probability of being selected in probabilistic sampling, which can be supported by random sampling [Tho14]. There are three probabilistic sampling techniques [HATB95]: simple random sampling, clustered sampling, and stratified sampling.

Survey instrument design phase. This phase concerns the rigorous design of the survey questionnaire and the development of the internal and survey questions [LSMdMH15]. While the internal questions represent open-ended questions that are later transformed into survey questions, the internal questions represent the research's primary goals [LSMdMH15]. To obtain solid results from the survey research, the design of the survey questionnaire is of utmost importance. Kasunic [Kas05] suggests several factors that should be considered when designing the survey questionnaire, e.g., determining the data to be measured, selecting the questionnaire type, selecting the execution method, and questionnaire length.

Survey instrument evaluation phase. After the survey instrument has been designed, this phase involves evaluating it to determine whether it has any deficiencies [LSMdMH15]. The evaluation can be done by conducting a pretest or a pilot survey. The survey evaluation primarily assesses the survey for clarity, understandability, acceptability, effectiveness, reliability, and validity [KPP+02, RP14]. A survey instrument can be evaluated using expert interviews, focus groups, pilot surveys, cognitive interviews, or experiments [PCL+04].

Survey data analysis phase. This phase comprises the analysis of the obtained survey data [LSMdMH15]. The data analysis depends on the type of questions used in the survey. Common methods to analyze the results of open-ended questions include content analysis, discourse analysis, grounded theory, phenomenology, and thematic analysis [GPRN18]. Common methods to analyze the results of closed-ended questions include statistical analysis, hypothesis testing, and data visualization [WRH+12]. In terms of the analysis process, Kitchenham et al. [KPP+02] suggest the following activities:

- *Data validation* involves checking the consistency and completeness of answers before analyzing survey results.
- *Partitioning of responses* comprises partitioning replies into subgroups prior to data analysis, e.g., through demographic questions.
- *Data coding* entails converting character string answers to nominal and ordinal scale data.

Drawing conclusions phase. This phase concludes the analysis of the survey data and presentation of the results [LSMdMH15]. The entire survey process must be evaluated and reviewed from a critical perspective. As part of the critical reflection, the two notions of validity and reliability must be addressed to understand the thoroughness and trustworthiness of the survey [LSMdMH15]. While the validity in its broadness refers to whether the questionnaire measured what it was intended to measure, reliability refers to whether the results can be generalized. The main motive is to identify potential threats early and reduce them. Research design decisions can completely mitigate threats while other threats remain open or partially reduced [GPRN18].

Documenting and reporting phase. The last phase is concerned with documenting the survey process and reporting the findings [LSMdMH15]. Documenting the survey process helps to enhance the survey's acceptance and quality. The documentation begins with specifying the research objectives and is iteratively updated as the research process continues. The various elements of the documentation should include research objectives and research questions, activity planning, sample method design, data collection methods, and data analysis methods [LSMdMH15]. Reporting the results might vary depending on the expectations and interests of the target audience. Because the target audience's interests may differ, Kasunic [Kas05] proposes conducting an audience analysis. Although the structure of the report is dependent on the target audience, the following seven topics should be considered for inclusion [CLVB03]: (i) abstract and/or executive summary, (ii) research objectives and problem statement, (iii) methodology and survey process, (iv) findings from data analysis, (v) discussion of results, threats to validity, and reliability, (vi) conclusions and acknowledgments, and (vii) eventual appendices.

In P3, we use a non-probabilistic, web-based survey to ask creators of scaling frameworks to study the reasons for their creation and the benefits and challenges related to their adoption.

3.2.4. Case Study Research

Case study research is one of several forms of social science research. A case study investigates a contemporary phenomenon, i.e., "*the case*", in its real-world context [Yin15]. The distinctive need for case study research stems from the desire to understand contemporary and complex social phenomena [BGM87, ESSD08]. Case study research enables researchers to focus on a "*case*" and maintain a holistic and real-world perspective [Yin15]. Case studies are employed in numerous situations to advance the knowledge of individual, group, organizational, social, political, and related phenomena. Case studies are commonly applied in areas like psychology, sociology, political science, and anthropology [Yin15]. Case study research is likewise well suited for software engineering research since the objects of study are contemporary phenomena that are difficult to study in isolation [RH09]. Case studies can be classified into four types [RH09]:

- *Exploratory case studies* aim to discover what is happening, gain new insights, and develop ideas and hypotheses for further research.
- *Descriptive case studies* seek to portray a situation or phenomenon if the generality of the situation or phenomenon is of secondary importance.
- *Explanatory case studies* strive to explain a situation or a problem, mostly but not necessarily in the form of a causal relationship.
- *Improving case studies* try to enhance a specific aspect of the studied phenomenon, similar to action research.

To inquire about a contemporary phenomenon, Yin [Yin15] proposes an iterative six-step approach (see Figure 3.4), described in the following.

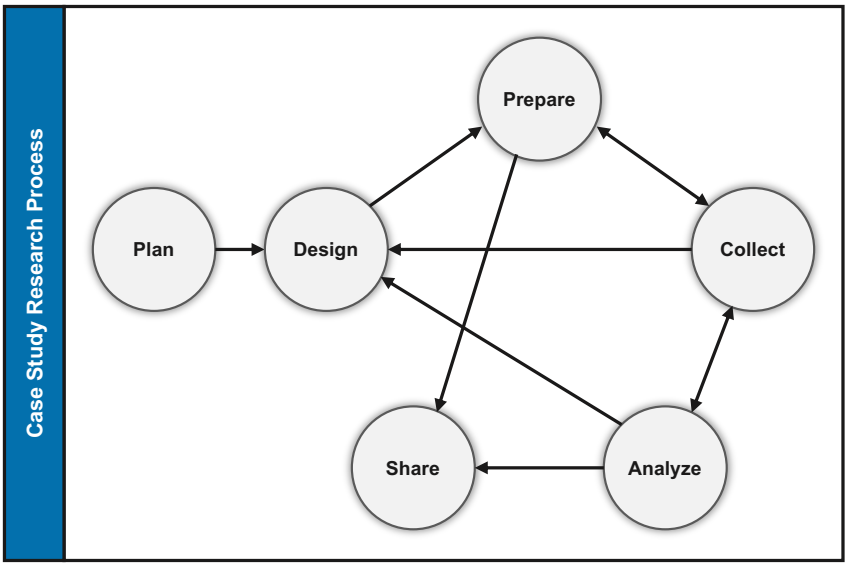


Figure 3.4.: Case study research approach [Yin15]

Planning phase. The planning phase is about evaluating the appropriateness of case study research, i.e., determining the rationale for conducting a case study, choosing the case study

method over other research approaches, and understanding its strengths and limitations [Yin15]. Case study research is appropriate when (i) a natural setting or a focus on contemporary events is needed, (ii) a solid theoretical base does not support the research phenomena, (iii) a rich natural setting can be fertile ground for generating theories, and (iv) subjects or events must not be controlled or manipulated in the course of a research project [BGM87]. According to Runeson and Höst [RH09] and Yin [Yin15], good planning for a case study and defining the research objectives are crucial to the entire research project. Any case study should begin with a comprehensive literature review and carefully consider the research questions and study objectives. A plan for a case study should include at least the following elements: an objective, the case, related theories, research questions, collection methods, and a selection strategy [Rob02].

Design phase. This phase is concerned with activities related to the research design, which include: (i) defining the unit(s) of analysis and the case(s) to be studied, (ii) developing and articulating theories and propositions, (iii) identifying issues underlying the anticipated study, (iv) determining the case study design, and (v) developing procedures to maintain the case study quality [Yin15]. The research design can be considered a "*blueprint*" for the research project. It links the research questions to the research conclusions through the steps undertaken during data collection and analysis [Yin15]. As shown in Figure 3.5, there are four types of specific designs for case studies: (i) single-case with a single unit of analysis (holistic), (ii) single-case with multiple units of analysis (embedded), (iii) multiple-case with a single unit of analysis (holistic), and (iv) multiple-case with multiple units of analysis (embedded).

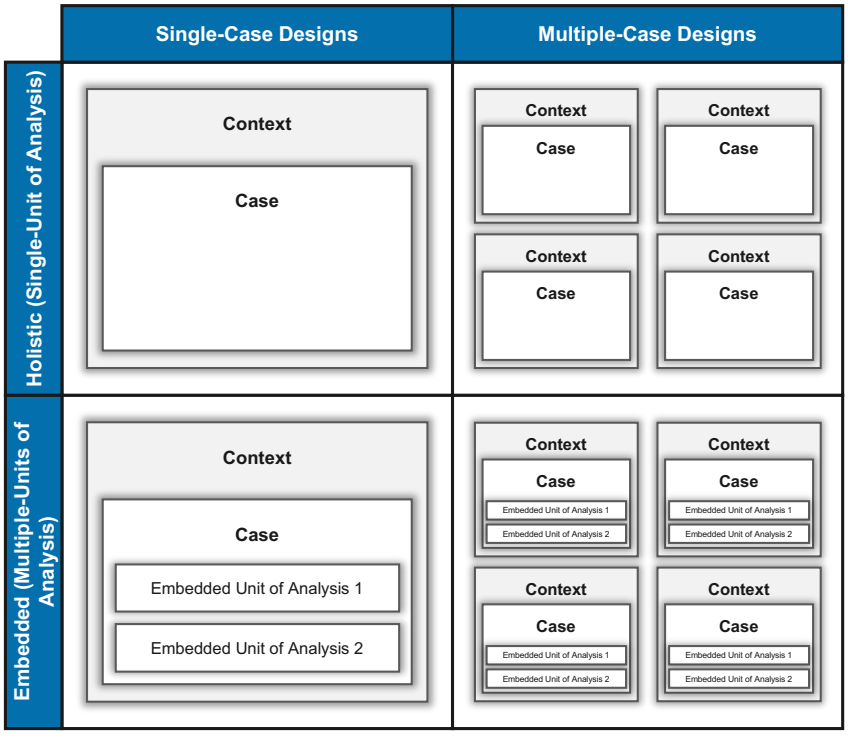


Figure 3.5.: Types of case study designs [Yin15]

Preparation phase. The preparation phase is concerned with preparing the case study, includ-

ing procedures for protecting human subjects, developing a case study protocol, and obtaining relevant approvals [Yin15]. The preparation phase also comprises determining the data sources for the case and potential embedded unit(s) of analysis, and preparing the data collection process [Yin15]. Even though the context of the phenomenon is defined, researchers must also decide on the data sources that will help answer the research questions. Decisions must be made about which people, places, events, and documents to observe, interview, or collect [Sta95a].

Data collection phase. The main objective of the data collection phase is to gather evidence from various data sources [Yin15]. Yin [Yin15] distinguishes between six sources: documents, archival records, interviews, direct observations, participant observations, and physical artifacts. To take advantage of these six sources, Yin [Yin15] proposes adhering to four principles of data collection: (i) using multiple sources of evidence, (ii) creating a case study database, (iii) maintaining a chain of evidence, and (iv) carefully using data from electronic sources. Regarding the first principle, Runeson and Höst [RH09] recommend using multiple data sources to limit the impact of an interpretation of a single data source and verify the repeatability of observations within the case, referred to as data triangulation [Fli92, Sta95a]. According to Yin [Yin15], the case study database compiles all collected documents and materials, e.g., audio recordings, interview protocols, and internal documents. To ensure a proper chain of evidence, Yin [Yin15] suggests using citations and footnotes in the report to refer to the relevant sources and maintaining a link between the protocol questions and the original study questions.

Data analysis phase. This phase deals with examining, categorizing, tabulating, testing, or otherwise recombining evidence to draw empirically-based conclusions while at the same time ensuring the rigor of the analysis [MH94, Yin15]. To achieve these goals and reduce potential analytical difficulties, Yin [Yin15] distinguishes four general analytic strategies. The first strategy, theoretical propositions, deals with theoretical propositions that guide the case study's analysis and help explain contextual conditions and explanations. The second strategy, working the data from the ground up, encourages to "*play with the data*" to identify patterns for describing the phenomenon of interest. The third strategy, developing a case description, aims to organize the case according to a descriptive framework. This strategy can be seen as complementary to other strategies by explaining the context and mechanisms of the phenomenon. The fourth strategy, examining plausible rival explanations, tries to define and test plausible rival explanations, which works in combination with the previous three strategies [Yin15]. Besides these four strategies, there are also five main analyzing techniques for case studies: pattern matching, explanation building, time-series analysis, logic models, and cross-case synthesis that can be applied to ensure the reliability of the findings [Yin15].

Sharing phase. The sharing phase is about publishing the case study findings based on the demand and expectations of the targeted audience [Yin15]. For adequate reporting, Walsham [Wal06] and Locke et al. [LSS13] recommend identifying the target audience of the case study report, developing the case study's compositional structure, and creating drafts of the report that peers and case study participants should review. Regardless of the form of the report, the compilation of the case study should include textual and visual materials and an ample case description, embedding the reader into the context of the phenomenon and displaying enough evidence for the reader to draw his or her conclusions [Yin15].

In P4, we use an embedded single-case design to compare different occurrences of LeSS adoption

inside a German automobile manufacturer, with the four investigated products representing the multiple units of analysis. In P8, we employ a holistic single-case design to investigate the collaboration between architects and agile teams within a large-scale agile development program of a consumer electronics retailer. Here, the program of the case company represents the single unit of analysis. In P9 and P10, we use a holistic multiple-case design to explore the expectations of enterprise architects and their role in supporting large-scale agile development endeavors within five German companies. The endeavors under study represent the single unit of analysis for each case company.

3.2.5. Pattern-Based Design Research

Design science approaches aim to create novel artifacts, i.e., solutions to relevant problems [HMPR04, PTRC07]. Researchers designing artifacts need to account for two critical criteria: rigor and relevance [HMPR04, RV08]. On the one hand, rigor can be achieved by using sound methods [BW96]. On the other hand, relevance can be obtained by considering the needs of the practitioners using these artifacts. In this sense, balancing both rigor and relevance remains a challenging task [Sta95b]. As early-stage design artifacts, patterns enable researchers to develop innovative artifacts that solve current and anticipated problems of practitioners in an organizational context. The creation of the patterns must also rely on sound methodologies and close collaboration with one or more industry partners to ensure their rigor and relevance [BMSS13a, BMSS13b]. Buckl et al. [BMSS13a] propose the iterative PDR method that balances rigor and relevance in creating patterns as early design artifacts. The PDR method consists of four phases (see Figure 3.6) and is described in the following.

Observation and conceptualization phase. In this phase, good practices from the industry are observed and documented following a typical pattern structure. At a minimum, the researcher describes the following concepts [BMSS13a]: the *problem* to be addressed, the *solution* that has proven to work well, the *context* in which the solution can be applied, and the *forces* that frame the solution space. The researcher can refine the documentation of the pattern candidates with the industry partner if required [BMSS13a].

Theory building and nexus instantiation phase. In this phase, the pattern candidates evolve into a pattern through three successful known uses, i.e., by fulfilling the *rule of three* [Cop96]. Here, the researcher integrates the new patterns into the organized pattern collection, i.e., pattern language, by defining relationships to the already existing patterns [BMSS13a].

Solution design and application phase. This phase is concerned with constructing a situated design artifact that can be applied in the context of an organization [BMSS13a]. In doing so, the solution design must be configured and adapted to the terminology of the applying organization, resulting in a so-called *configured solution design*. The resulting configured design serves as a blueprint implemented as a new solution by the organization under consideration [BMSS13a, BMSS13b]. The actual implementation of the configured solution design is referred to as an *instantiated solution design* [BMSS13b].

Evaluation and learning phase. While time passes, the instantiated solution design may be subject to evolution. For example, the organization may find a better solution over time, result-

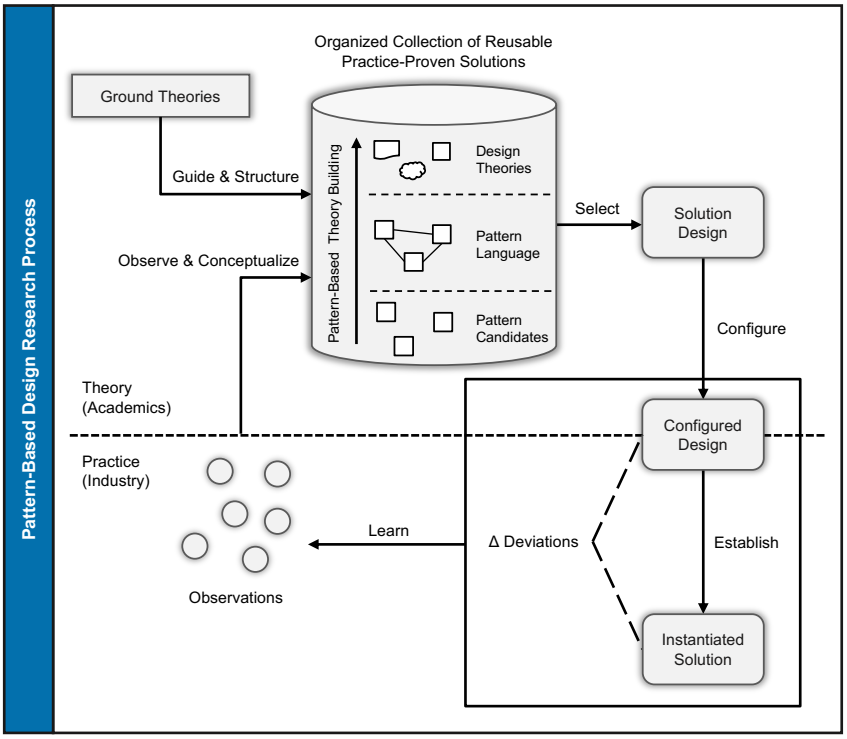


Figure 3.6.: Pattern-based design research approach [BMSS13a]

ing in potential deviations from the initially configured solution design [BMSS13a, BMSS13b]. These deviations represent the artifact's evolution and can be ascribed to the ongoing change in the organization's environments, contexts, and goals [BMSS13a]. Hence, the evaluation and learning phase aims to detect and document deviations between the actual and initially instantiated solution design. These deviations can be used as a basis to extend the existing knowledge basis or identify new best practices [BMSS13a, BMSS13b]. Thereby, two main types of deviations can be distinguished, leading to different learning types [BMSS13a]:

- Deviations in the instantiated solution design representing minor changes from the configured design can be attributed to the corresponding organizational context or problem.
- Major changes in the instantiated solution design not matching the configured solution's organizational context or problem represent newly observed best practices.

When new best practices are observed, they must be documented as pattern candidates, as described in the observe and conceptualize phase. Minor changes usually lead to revising existing patterns related to their documentation and relationships to other patterns [BMSS13a].

In P6, we use the PDR approach to observe and conceptualize a pattern language consisting of patterns and concepts that can be used to address typical concerns of stakeholders in large-scale agile development. In P7, we employ the same approach to identify typical concerns of agile coaches and scrum masters in large-scale agile development and propose patterns and concepts that can be applied to solve their concerns.

Part B

1. Revealing the State of the Art of Large-Scale Agile Development Research: A Systematic Mapping Study

Table 4.1. Fact sheet publication P1

Authors	Uludağ, Ömer* Philipp, Pascal* Putta, Abheeshta Paasivaara, Maria Lassenius, Casper Matthes, Florian* *Technische Universität München, Chair of Software Engineering for Business Information Systems, Boltzmannstraße 3, D-85748 Garching, Germany
Outlet	Journal of Systems and Software (JSS)
Status	Published
Contribution of first author	Problem definition, research design, data collection, data analysis, interpretation, reporting

Abstract. *Context:* Success with agile methods in the small scale has led to an increasing adoption also in large development undertakings and organizations. Recent years have also seen an increasing amount of primary research on the topic, as well as a number of systematic literature reviews. However, there is no systematic overview of the whole research field. *Objective:* This work identifies, classifies, and evaluates the state of the art of research in large-scale agile development. *Method:* We conducted a systematic mapping study and rigorously selected 136 studies. We designed a classification framework and extracted key information from the studies. We synthesized the obtained data and created an overview of the state of the art. *Results:* This work contributes with (i) a description of large-scale agile endeavors reported in the industry, (ii) a systematic map of existing research in the field, (iii) an overview of influential studies, (iv) an overview of the central research themes, and (v) a research agenda for future research. *Conclusion:* This study portrays the state of the art in large-scale agile development and offers researchers and practitioners a reflection of the past thirteen years of research and practice on the large-scale application of agile methods.

2. Investigating the Role of Architects in Scaling Agile Frameworks

Table 4.2. Fact sheet publication P2

Authors	Uludağ, Ömer* Kleehaus, Martin* Xu, Xian* Matthes, Florian* *Technische Universität München, Chair of Software Engineering for Business Information Systems, Boltzmannstraße 3, D-85748 Garching, Germany
Outlet	21st International Enterprise Distributed Object Computing Conference (EDOC)
Status	Published
Contribution of first author	Problem definition, research design, data collection, data analysis, interpretation, reporting

Abstract. This study describes the roles of architects in scaling agile frameworks with the help of a structured literature review. We aim to provide a primary analysis of 20 identified scaling agile frameworks. Subsequently, we thoroughly describe three popular scaling agile frameworks: Scaled Agile Framework, Large Scale Scrum, and Disciplined Agile 2.0. After specifying the main concepts of scaling agile frameworks, we characterize roles of enterprise, software, solution, and information architects, as identified in four scaling agile frameworks. Finally, we provide a discussion of generalizable findings on the role of architects in scaling agile frameworks.

3. Evolution of the Agile Scaling Frameworks

Table 4.3. Fact sheet publication P3

Authors	Uludağ, Ömer* Putta, Abheeshta Paasivaara, Maria Matthes, Florian* *Technische Universität München, Chair of Software Engineering for Business Information Systems, Boltzmannstraße 3, D-85748 Garching, Germany
Outlet	22nd International Conference on Agile Software Development (AGILE)
Status	Published
Contribution of first author	Problem definition, research design, data collection, data analysis, interpretation, reporting

Abstract. Over the past decade, agile methods have become the favored choice for projects undertaken in rapidly changing environments. The success of agile methods in small, co-located projects has inspired companies to apply them in larger projects. Agile scaling frameworks, such as Large Scale Scrum and Scaled Agile Framework, have been invented by practitioners to scale agile to large projects and organizations. Given the importance of agile scaling frameworks, research on those frameworks is still limited. This paper presents our findings from an empirical survey answered by the methodologists of 15 agile scaling frameworks. We explored (i) framework evolution, (ii) main reasons behind their creation, (iii) benefits, and (iv) challenges of adopting these frameworks. The most common reasons behind creating the frameworks were improving the organization's agility and collaboration between agile teams. The most commonly claimed benefits included enabling frequent deliveries and enhancing employee satisfaction, motivation, and engagement. The most mentioned challenges were using frameworks as cooking recipes instead of focusing on changing people's culture and mindset.

4. Investigating the Adoption and Application of Large-Scale Scrum at a German Automobile Manufacturer

Table 4.4. Fact sheet publication P4

Authors	Uludağ, Ömer* Kleehaus, Martin* Dreymann, Niklas* Kabelin, Christian Matthes, Florian* *Technische Universität München, Chair of Software Engineering for Business Information Systems, Boltzmannstraße 3, D-85748 Garching, Germany
Outlet	14th International Conference on Global Software Engineering (ICGSE)
Status	Published
Contribution of first author	Problem definition, research design, data collection, data analysis, interpretation, reporting

Abstract. Over the last two decades, agile methods have been adopted by an increasing number of organizations to improve their software development processes. In contrast to traditional methods, agile methods place more emphasis on flexible processes than on detailed upfront plans and heavy documentation. Since agile methods have proved to be successful at the team level, large organizations are now aiming to scale agile methods to the enterprise level by adopting and applying so-called scaling agile frameworks such as Large-Scale Scrum (LeSS) or Scaled Agile Framework (SAFe). Although there is a growing body of literature on large-scale agile development, literature documenting actual experiences related to scaling agile frameworks is still scarce. This paper aims to fill this gap by providing a case study on the adoption and application of LeSS in four different products of a German automobile manufacturer. Based on seven interviews, we present how the organization adopted and applied LeSS, and discuss related challenges and success factors. The comparison of the products indicates that transparency, training courses and workshops, and change management are crucial for a successful adoption.

5. Identifying and Structuring Challenges in Large-Scale Agile Development Programs based on a Structured Literature Review

Table 4.5. Fact sheet publication P5

Authors	Uludağ, Ömer* Kleehaus, Martin* Caprano, Christoph* Matthes, Florian* *Technische Universität München, Chair of Software Engineering for Business Information Systems, Boltzmannstraße 3, D-85748 Garching, Germany
Outlet	22nd International Conference on Enterprise Distributed Object Computing (EDOC)
Status	Published
Contribution of first author	Problem definition, research design, data collection, data analysis, interpretation, reporting

Abstract. Over the last two decades, agile methods have transformed and brought unique changes to software development practice by strongly emphasizing team collaboration, customer involvement, and change tolerance. The success of agile methods for small, co-located teams has inspired organizations to increasingly apply agile practices to large-scale efforts. Since these methods are originally designed for small teams, unprecedented challenges occur when introducing them at larger scale, such as inter-team coordination and communication, dependencies with other organizational units or general resistances to changes. Compared to the rich body of agile software development literature describing typical challenges, recurring challenges of stakeholders and initiatives in large-scale agile development has not yet been studied through secondary studies sufficiently. With this paper, we aim to fill this gap by presenting a structured literature review on challenges in large-scale agile development. We identified 79 challenges grouped into eleven categories.

6. Documenting Recurring Concerns and Patterns in Large-Scale Agile Development

Table 4.6. Fact sheet publication P6

Authors	Uludağ, Ömer* Harders, Nina* Matthes, Florian* *Technische Universität München, Chair of Software Engineering for Business Information Systems, Boltzmannstraße 3, D-85748 Garching, Germany
Outlet	24th European Conference on Pattern Languages of Programs (EPLoP)
Status	Published
Contribution of first author	Problem definition, research design, data collection, data analysis, interpretation, reporting

Abstract. The introduction of agile methods at scale entails unique concerns such as inter-team coordination, dependencies to other organizational units, or distribution of work without a defined architecture. Compared to the rich body of agile software development literature describing typical challenges and best practices, recurring concerns and patterns in large-scale agile development are not yet documented extensively. We aim to fill this gap by presenting a pattern language for large-scale agile software development as part of our larger research initiative in close collaboration with 10 companies. The structure and practical relevance of the proposed language were evaluated by 14 interviews. In this paper, we showcase our pattern language by presenting four patterns.

7. Identifying and Documenting Recurring Concerns and Best Practices of Agile Coaches and Scrum Masters in Large-Scale Agile Development

Table 4.7. Fact sheet publication P7

Authors	Uludağ, Ömer* Matthes, Florian* *Technische Universität München, Chair of Software Engineering for Business Information Systems, Boltzmannstraße 3, D-85748 Garching, Germany
Outlet	26th International Conference on Pattern Languages of Programs (PLoP)
Status	Published
Contribution of first author	Problem definition, research design, data collection, data analysis, interpretation, reporting

Abstract. Ever since the release of the agile manifesto in 2001, agile methods have received widespread interest in industry and academia. Agile methods have transformed and brought unique changes to software development practices by strongly emphasizing team collaboration, change tolerance, and active customer involvement. Their proven benefits have also inspired organizations to apply them in large-scale settings. However, the adoption of agile methods at scale entails unique challenges such as coordinating and aligning multiple large-scale agile activities, dealing with internal silos, and establishing an agile culture & mindset throughout the organization. In particular, agile coaches and scrum masters are confronted with unprecedented concerns in large-scale agile development. Notwithstanding their importance for large-scale agile endeavors, extant literature still lacks an overview of their typical concerns and a collection of patterns to address them. Against this backdrop, we provide an overview of typical concerns and present five best practices of agile coaches and scrum masters in large-scale agile development.

8. Using Social Network Analysis to Investigate the Collaboration Between Architects and Agile Teams: A Case Study of a Large-Scale Agile Development Program in a German Consumer Electronics Company

Table 4.8. Fact sheet publication P8

Authors	Uludağ, Ömer* Kleehaus, Martin* Erçelik, Soner* Matthes, Florian* *Technische Universität München, Chair of Software Engineering for Business Information Systems, Boltzmannstraße 3, D-85748 Garching, Germany
Outlet	20th International Conference on Agile Software Development (AGILE)
Status	Published
Contribution of first author	Problem definition, research design, data collection, data analysis, interpretation, reporting

Abstract. Over the past two decades, agile methods have transformed and brought unique changes to software development practice by strongly emphasizing team collaboration, customer involvement, and change tolerance. The success of agile methods for small, co-located teams has inspired organizations to increasingly use them on a larger scale to build complex software systems. The scaling of agile methods poses new challenges such as inter-team coordination, dependencies to other existing environments or distribution of work without a defined architecture. The latter is also the reason why large-scale agile development has been subject to criticism since it neglects detailed assistance on software architecting. Although there is a growing body of literature on large-scale agile development, literature documenting the collaboration between architects and agile teams in such development efforts is still scarce. As little research has been conducted on this issue, this paper aims to fill this gap by providing a case study of a German consumer electronics retailer's large-scale agile development program. Based on social network analysis, this study describes the collaboration between architects and agile teams in terms of architecture sharing.

9. What to Expect from Enterprise Architects in Large-Scale Agile Development? A Multiple-Case Study

Table 4.9. Fact sheet publication P9

Authors	Uludağ, Ömer* Kleehaus, Martin* Matthes, Florian* *Technische Universität München, Chair of Software Engineering for Business Information Systems, Boltzmannstraße 3, D-85748 Garching, Germany
Outlet	25th Americas Conference on Information Systems (AMCIS)
Status	Published
Contribution of first author	Problem definition, research design, data collection, data analysis, interpretation, reporting

Abstract. In modern times, traditional enterprises are confronted with rapidly changing customer demands, increasing market dynamics, and continuous emergence of technological advancements. Confronted with the imperatives of a digital world, companies are striving to adopt agile methods on a larger scale to meet these requirements. In recent years, enterprise architecture management has established itself as a valuable governance mechanism for coordinating large-scale agile transformations by connecting strategic considerations to the execution of transformation projects. Our research is motivated by the lack of empirical studies on the collaboration between enterprise architects and agile teams. Against this backdrop, we present a multiple-case study of five leading German companies that aims to shed light on this field of tension. Based on our results from 20 semi-structured interviews, we present the expectations of agile teams for enterprise architects and how they are fulfilled.

10. Investigating the Role of Enterprise Architects in Supporting Large-Scale Agile Transformations: A Multiple-Case Study

Table 4.10. Fact sheet publication P10

Authors	Uludağ, Ömer* Matthes, Florian* *Technische Universität München, Chair of Software Engineering for Business Information Systems, Boltzmannstraße 3, D-85748 Garching, Germany
Outlet	26th Americas Conference on Information Systems (AMCIS)
Status	Published
Contribution of first author	Problem definition, research design, data collection, data analysis, interpretation, reporting

Abstract. In today's competitive environments, companies must cope with changing customer demands, regulatory uncertainties, and new technological advances. To this end, companies increasingly undergo large-scale agile transformations to meet these requirements. In recent years, enterprise architecture management has established itself as a valuable governance mechanism for coordinating large-scale agile transformations by connecting strategic considerations to the execution of transformation projects. Empirical studies investigating the role of enterprise architects (EAs) in this context are still scarce. We present a multiple-case study of five major German companies that aims to shed light on the role of EAs in supporting large-scale agile transformations. Based on our results from eighteen interviews, we present a set of typical responsibilities of EAs. We also describe the expectations of various stakeholders towards EAs and the challenges they face.

Part C

CHAPTER 4

Discussion

This chapter discusses the key results of the embedded publications (see Section 4.1) and describes the implications of the results for research and practice (see Sections 4.2 and 4.3).

4.1. Summary of Results

We published ten research articles to substantially contribute toward answering the four research questions formulated in Section 1.2. We present the main findings of the ten publications alongside the research questions and relate them to the existing literature.

Key findings related to RQ1. Over the past years, many studies on large-scale agile development have appeared in scientific conferences and journals, leading to a large body of knowledge [UPP+22]. Although several secondary studies were published (cf. [DPL16, EWC21]), none provide an overview of the entire research field. Against this research gap, we formulated the first research question as follows:

> Research question 1 (RQ1)
>
> What is the state of the art in large-scale agile development research?

To answer the first research question, we conducted a systematic mapping study, covering 136 publications from 2007 to 2019 (see P1). In the scope of this research question, we (i) analyzed what has been referred to as "*large-scale agile development*", (ii) examined publication trends and characteristics of existing research, (iii) revealed seminal works, and (iv) identified central research streams and open research gaps.

Although several researchers have made an initial attempt to define the term "*large-scale agile development*" (cf. [DFI14, DPL16, FH18]), there is some ambiguity about the meaning of this term, inhibiting effective collaboration and progress in the research area [DFI14]. Departing from this issue, we examined the reported case companies and analyzed their endeavors that were described as "*large-scale agile development*" (see P1).

We identified 158 companies reported in the existing literature, of which 137 were anonymized, and 21 were explicitly mentioned. Ericsson was the most frequently studied company, followed by the Norwegian Public Service Pension Fund (see P1). Although the reported case companies were spread across the globe, we found several companies coming from Europe and North America. This observation is consistent with Digital.ai's State of Agile survey [Dig20], confirming a topic concentration in Europe and North America (see P1). While the reported companies came from ten different sectors, almost a third were from the Information Technology (IT) sector, followed by the financial and public sectors. Again, our findings are in line with Digital.ai's State of Agile survey [Dig20], stating that most companies adopting (large-scale) agile methods come from the IT and financial sectors (see P1). While most of the reported companies had more than 20,000 employees, we observed that companies with fewer than 1,000 employees also adopt agile methods at scale. Our findings and Digital.ai's survey results [Dig20] indicate that companies adopt agile methods at scale, regardless of their size. However, our results show that most companies have more than 5,000 employees, accounting for nearly 70% of all identified companies. This phenomenon is likely more relevant for large companies than for small companies (see P1).

To understand what has been reported as "*large-scale agile development*", we used the classification by Fuchs and Hess [FH18] to categorize the reported large-scale agile development efforts. Almost half of all companies applied agile practices as a whole, i.e., "*organizational agility*", followed by companies using agile methods in large multi-team settings, i.e., "*large agile multi-team settings*" (see Section 2.3.1). While both categories were reported in four companies, we could not determine the type for 16 companies due to superficial context information (see P1).

We revealed 110 development efforts with multiple agile teams. We uncovered several companies with multiple efforts, such as SAP having five large agile multi-team settings involving between 4 and 13 teams (see P1). Most reported settings were large, employing between 11 and 100 people or 2 to 9 teams, followed by very large settings, involving between 101 and 8,000 people or 10 to 40 teams. When comparing our results with Digital.ai's survey findings [Dig20], we found several differences: most settings we identified had fewer than 100 people. In contrast, the survey indicated about a third of their respondents with the same size. While 36% of State of Agile survey respondents reported sizes between 101 and 1,000 employees [Dig20], this accounted for 46.67% of agile multi-team settings from our study. While 31% of the survey respondents indicated sizes greater than 1,000 employees [Dig20], we identified only one very large agile multi-team setting at Cisco with 8,000 employees (see P1).

To acquire a more in-depth understanding of what the authors of the selected studies meant by "*scaling agile methods*", we examined the case descriptions for possible scaling and complexity factors (see P1). As a result, we identified six factors (see Figure 4.1). Organizational size, i.e., the number of agile teams working together or the number of people involved in the development effort, was the most frequently cited factor, followed by organizational distribution, i.e., number of sites or geographic distribution. The complexity of the developed systems, i.e., product size

(measured in lines of code), number of subsystems, and number of requirements and features, was the third most reported factor, followed by the number of companies working together, the number of customers, and the available budget (see P1). Ambler [Amb07] has already provided some scaling and complexity factors. In accordance with Ambler [Amb07], we believe that organizational size is the primary scaling and complexity factor for characterizing the term "*scaling agile methods*". Like Ambler [Amb07], we are also convinced that organizational distribution and system complexity are likewise important factors to consider. In addition to Ambler [Amb07], we uncovered the number of collaborating companies, the number of customers, and the available budget for development efforts as further factors (see P1).

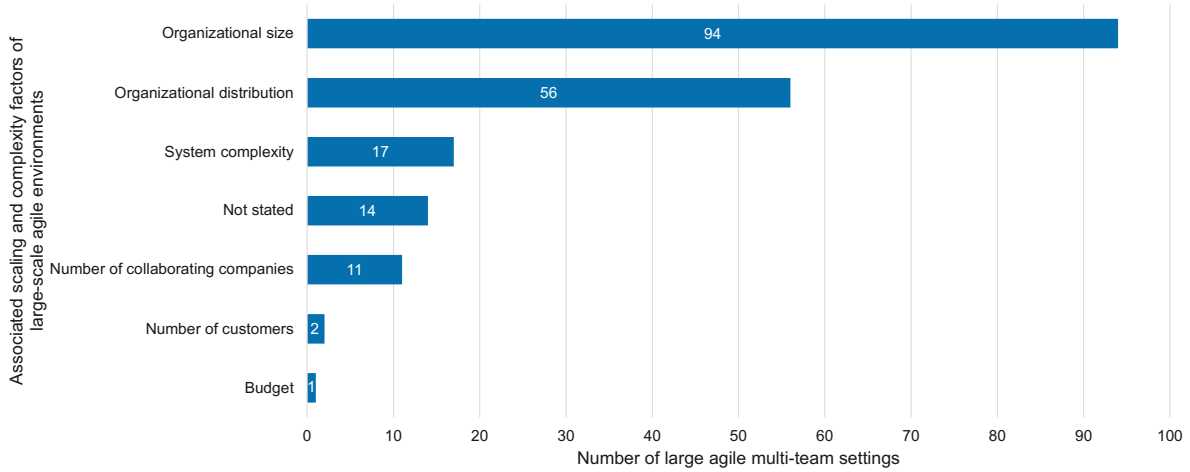


Figure 4.1.: Scaling and complexity factors (based on P1)

To reveal potential publication trends and characteristics, we selected a set of variables that focused on each study's publication and bibliographic data, e.g., publication year, contribution type, and research approach (see P1). Below, we detail the key facts drawn from our analysis.

Figure 4.2 indicates increasing attention from the research community on the topic, as the number of studies has been steadily increasing with slight fluctuations (see P1). In particular, after 2013, almost 85% of the studies were published. The increasing trend on the topic culminated in 2018 and 2019, when publications doubled compared to previous years (see P1). A possible explanation for this observation could be the initiation of two workshops contributing to large-scale agile development research, namely the International Workshop on Autonomous Agile Teams in 2018 and the International Workshop on Agile Transformation in 2019.

A total of 22 states worldwide contributed to research, of which the most were from Europe, accounting for almost 90% of all publications. The research theme received considerable interest from Scandinavia, i.e., Finland, Norway, and Sweden, followed by Germany.

We investigated each study's publication types, venues, and domains to examine pertinent publication channels for large-scale agile development research. Almost 50% of the studies were published in conferences, followed by journals. Such a high number of journal and conference papers may indicate that the research field matures (see P1). While the prominent systematic literature review by Dikert et al. [DPL16] in 2016 mainly included experience and opinion re-

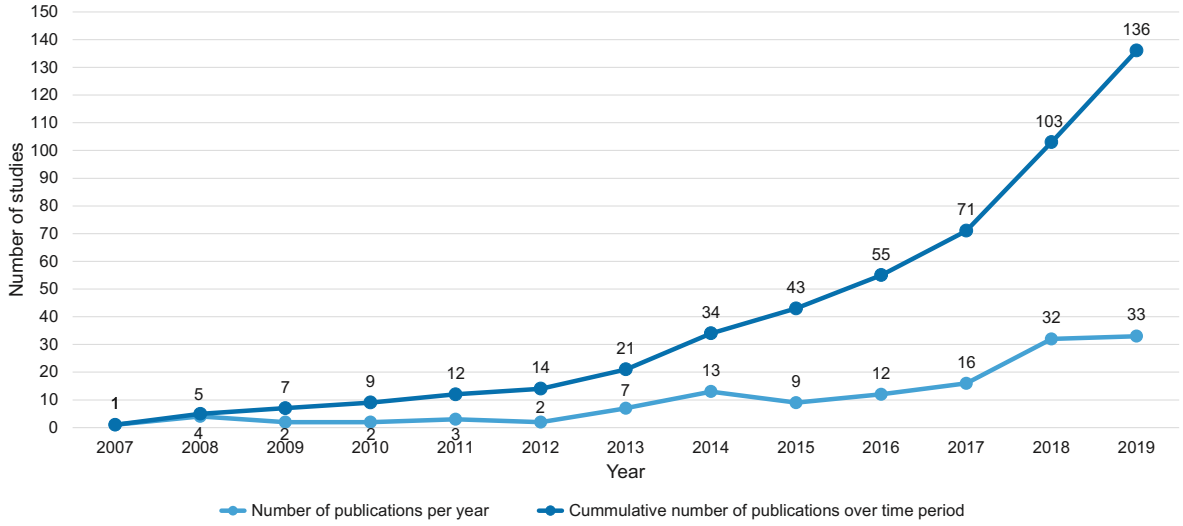


Figure 4.2.: Publications on large-scale agile development over time (based on P1)

ports from practitioners with a lack of scientific rigor, nearly 80% of the studies we selected were published in scientific journals and conferences, signaling a shift in the scientific basis of the research field. We believe this academic traction is triggered by the ubiquitous industrial relevance of the topic, requiring scientific assistance by researchers (cf. [DPL16, DFP19]). Research on large-scale agile development was published in eight different domains. Unsurprisingly, most publications were covered by publication venues from the software engineering domain, followed by the information systems research discipline (see P1). Although most of the studies came from the software engineering field, we witnessed a growing interest from other research communities, e.g., information systems, project management, and enterprise computing (see P1). The 136 selected studies were published in 46 venues, including 17 journals, 23 conferences, and six workshops. The International Conference on Agile Software Development and the International Workshop on Large-Scale Agile Development are the top-2 venues (see P1). We also noted that several studies were also present in leading software engineering journals, e.g., Information and Software Technology and Empirical Software Engineering (see P1).

To characterize the research that has been conducted, we further investigated the research types, methods, research outcomes, and research data types (see P1). About 80% of the studies evaluated the large-scale adoption of agile methods in the industry, typically through exploratory case studies, mixed methods, surveys, or similar empirical methods. About 11% of the studies presented novel solutions or significant extensions of existing solutions identified in practice, followed by philosophical papers that framed the research field through conceptual frameworks or taxonomies (see P1). The fact that evaluation research is widely used in large-scale agile development research positively impacts transferring current research results to the industry. The prevalence of evaluation research studies is natural for phenomena such as large-scale agile development since it is mainly practice-driven (see P1). The majority of the selected publications were empirical and qualitative, including mainly case studies. About 85% of the studies contributed to lessons learned and guidelines and were observational, analyzing and describing

the industry's large-scale application of agile methods. While only 15% of the studies contributed models, frameworks/methods, and theories, none of the studies contributed to creating new or improved tools. Based on this finding, we conclude that further research is required to develop conceptual models and theories to strengthen the research field's theoretical foundations and create rigorously developed frameworks, methods, and tools to support practitioners (see P1). Comparing the selected studies with those from the agile software development research area (cf. [DNBM12]), we revealed that existing studies on large-scale agile development did not pay enough attention to establishing theoretical underpinnings. Analogous to the agile software development research field and agreeing with Dingsøyr et al. [DNBM12], we believe that the large-scale agile development research area can only mature if adequate efforts are made to provide a solid theoretical scaffold. Like Batra [Bat20], we also observed an apparent lack of quantitative studies that use surveys as data collection instruments to conduct quantitative investigations and assessments. We revealed that almost 90% of the publications were primary and the remainder secondary. We did not find any tertiary studies (see P1).

To better understand the intellectual structure of the research field, we identified seminal works as recommended by Dingsøyr et al. [DNBM12] and Nerur et al. [NRN08]. To this end, and similar to Herbold et al. [HATG21], we extracted data on citation counts from Google Scholar and considered studies with the top 10% citations as influential (see P1). Table 4.1 provides an overview of the identified seminal studies.

To assess the state of the art and maturity of the research field, we explored central research themes and existing research gaps. Using a systematic keywording process [PFMM08], we revealed clusters to identify pertinent research topics and categorized the selected studies according to the main research themes. We identified ten research streams (see P1):

- The *Agile architecture* research stream addresses how companies unite the at first glance conflicting topics of agility and architecture to develop complex products and how agile teams are architecturally aligned at the enterprise level.
- The *Agile planning* research stream provides scientific insights into how large-scale agile projects and organizations conduct agile planning activities.
- The *Agile portfolio management* research stream covers how companies tailor their traditional portfolio management approaches to agile environments.
- The *Agile practices at scale* research stream investigates how companies tailor genuine agile practices to fit large-scale projects.
- The *Communication and coordination* research stream tackles the issue of how agile teams effectively coordinate and communicate with each other.
- The *Global and distributed software engineering* research stream studies how companies apply agile methods in large, globally distributed projects.
- The *Large-scale agile transformations* research stream centers on how companies undertake these transformations to meet the imperatives of agile companies.
- The *Scaling agile frameworks* research stream concentrates on studying the introduction of scaling frameworks in organizations.

Table 4.1. Seminal studies in large-scale agile development (based on P1)

Title	Authors	Publication type	year	No. of citations	Citations per year
Challenges and success factors for large-scale agile transformations: a systematic literature review	Kim Dikert, Maria Paasivaara, Casper Lassenius	Journal	2016	273	68
A framework to support the evaluation, adoption and improvement of agile methods in practice	Asif Qumer, Brian Henderson-Sellers	Journal	2008	256	21
A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case	Kai Petersen, Claes Wohlin	Journal	2009	229	21
Agile methods rapidly replacing traditional methods at nokia: a survey of opinions on agile transformation	Maarit Laanti, Outi Salo, Pekka Abrahamsson	Journal	2011	227	25
The effect of moving from a plan-driven to an incremental software development approach with agile practices: an industrial case study	Kai Petersen, Claes Wohlin	Journal	2010	166	17
Agile portfolio management: an empirical perspective on the practice in use	Christoph J. Stettina, Jeanette Hörz	Journal	2015	126	25
Distributed agile development: using scrum in a large project	Maria Paasivaara, Sandra Durasiewicz, Casper Lassenius	Conference	2008	119	10
Using scrum in a globally distributed project: a case study	Maria Paasivaara, Sandra Durasiewicz, Casper Lassenius	Journal	2008	101	8
Communities of practice in a large distributed agile software development organization - case ericsson	Maria Paasivaara, Casper Lassenius	Journal	2014	98	16
Inter-team coordination in large- scale globally distributed scrum: do scrum-of-scrums really work?	Maria Paasivaara, Casper Lassenius, Ville T. Heikkilä	Conference	2012	91	11
A case study on benefits and side-effects of agile practices in large-scale requirements engineering	Elizabeth Bjarnason, Krzysztof Wnuk, Björn Regnell	Workshop	2012	87	10
Combining agile software projects and large-scale organizational agility	Petri Kettunen, Maarit Laanti	Journal	2008	86	7
Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation	Torgeir Dingsøyr, Nils B. Moe, Tor E. Fægri, Eva A. Seim	Journal	2018	83	42

- The *Taxonomy* research stream tackles the issue of how to create more conceptual clarity around large-scale agile development.
- The *Team autonomy* research stream deals with how complex organizational structures impact team autonomy and how companies strike a balance between self-organizing teams focused on their own goals and those of the broader organization.

While some research streams, i.e., taxonomy and agile portfolio management, received little interest, scholars intensively contributed to the agile practices at scale, communication and coordination, and scaling agile frameworks research streams (see P1). While early research started with contributions related to the agile practices at scale, global and distributed software engineering, and scaling agile frameworks research streams, we noted a recent interest by researchers in the team autonomy and large-scale agile transformations research streams (see P1).

Following Kitchenham et al. [KBB11], we derived an agenda for future research that scholar can build upon. Although researchers made considerable efforts to close research gaps in various research streams, we identified 81 research questions. We identified the most research questions in the communication and coordination research stream, followed by the agile architecture and agile planning research streams (see P1).

An overview of the number of studies assigned to research streams and the number of research questions identified in the research streams is shown in Figure 4.3 and Figure 4.4.

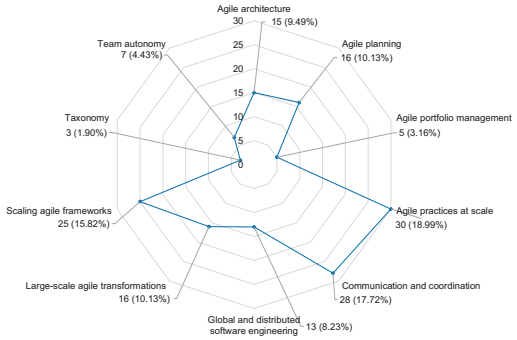


Figure 4.3.: Number studies per research stream (based on P1)

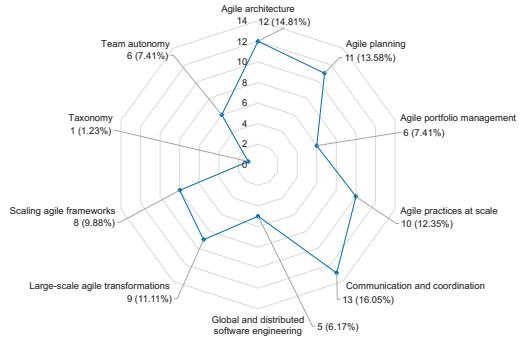


Figure 4.4.: Number research questions per research stream (based on P1)

Table 4.2 summarizes the key results of the embedded publications related to RQ1.

Key findings related to RQ2. Several scaling frameworks were proposed by consultants to assist practitioners in the adoption of agile practices in large-scale organizations and projects [Vai14, EWC21]. Despite this industrial interest [Dig21], empirical evidence on their adoption is still very much in its infancy [CC19a] since they neglect to provide (i) a comprehensive overview of the frameworks, (ii) first-hand information from their inventors, and (iii) report on the adoption of other frameworks besides SAFe. Motivated by this backdrop, we articulated the second research question as follows:

Research question 2 (RQ2)

What scaling agile frameworks exist, and what are the reasons, benefits, and challenges of adopting them?

To address the second research question, we conducted a structured literature review covering 146 publications from 2001 to 2017. We compiled a list of scaling frameworks and compare them using descriptive and maturity-related information (see P2). Using the list of the identified frameworks, we surveyed their inventors to learn about the main reasons for their development and the benefits and challenges of their usage (see P3). Finally, we conducted a single-case study on a German car manufacturer to study how LeSS was adopted (see P4).

In previous studies, several researchers attempted to compare scaling frameworks based on different criteria. For instance, Vaidya [Vai14] reviewed the three most widely known scaling frameworks: SAFe, LeSS, and DAD, by analyzing their recommended roles, processes, and other

Table 4.2. Overview of key findings related to RQ1

RQ	No.	Key findings

RQ1	P1	• Large-scale agile development refers to bringing agile practices to more people, even to entire companies
		  • The idea of adopting agile methods at scale permeates many companies of different sizes, almost on all continents and in all industries
		  • The term "large-scale agile development" entails the application of agile practices in organizations as a whole and/or their application in large multi-team settings
		  • While various scaling and complexity factors have been suggested to denote the term "scaling agile methods", organizational size, i.e., the number of agile teams or people involved in the development endeavors, is the primary factor for characterizing this term
		• The topic of large-scale agile development has been lavished attention by researchers worldwide
		  • Academic interest has steadily increased, while the number of publications started to accelerate in 2013
		  • While the topic is domicile in the software engineering domain, it met considerable interest in various publication channels and research disciplines
		  • The research field is mainly characterized by empirical and qualitative research, signaling an apparent lack of quantitative studies
		• Identification of 13 seminal studies on large-scale agile development
		  • Most seminal publications discuss implications, issues, benefits, and success factors associated with the large-scale adoption of agile practices in plan-driven organizations
		  • The systematic literature review by Dikert et al. [DPL16] is the most influential study
		• Research on large-scale agile development is clustered into ten research streams
		  • While early research started with contributions related to the agile practices at scale, global and distributed software engineering, and scaling agile frameworks research streams, we noted a recent interest by researchers in the team autonomy and large-scale agile transformations research streams
		  • Although researchers made considerable efforts to close research gaps in various research streams of large-scale agile development, a total of 81 research questions have been identified

salient features. Alqudah and Razali [AR16] contrasted DAD, LeSS, Nexus, RAGE, SAFe, and the Spotify Model (Spotify) based on various criteria, e.g., supported team sizes, available training and certificates, and underlying agile methods and practices. Since studies often examined varying frameworks and simple literature searches revealed additional frameworks, we attempted in P2 to obtain a complete overview of existing frameworks. The structured literature review identified 20 scaling frameworks, of which several were sparsely investigated in previous studies, e.g., Continuous Agile Framework and Enterprise Transition Framework. Using various criteria, e.g., the number of contributions citing them or the number of known companies adopting them, we assessed the maturity of the identified frameworks. While our results indicate that most frameworks had a low to medium maturity, the three most often discussed frameworks in the literature, i.e., SAFe, LeSS, and DAD, had the highest maturity (see P2). This finding aligns with Digital.ai's annual State of Agile survey [Dig20], stating that SAFe, LeSS, and DAD are among the most widely adopted frameworks.

To deepen our primary analysis of scaling frameworks in P2 and extend the existing literature that provides a one-sided analysis of frameworks mainly in the form of single-case studies (cf. [Paa17, Gus18]), we used the list of the identified frameworks to survey their creators (see P3). We extended the initial list of 20 scaling frameworks by additional two frameworks, namely HSD and Parallel, as their methodologists approached us during two agile conferences (see P3). A comprehensive overview of the 22 identified scaling frameworks is presented in Table 4.3[1].

[1]The names of scaling frameworks whose methodologists participated in our survey are set in bold.

Table 4.3. Overview of scaling agile frameworks (based on P2 and P3)

Framework	Methodologist	Organization	Publication date	Category	Scaling level	Maturity
Dynamic Systems Development Method Agile Project Framework for Scrum (DSDM)	Arie van Bennekum	DSDM Consortium	1997	Framework	Portfolio	◕
Crystal Family (Crystal)	Alistair Cockburn	-	1998	Set of methods	Team	◑
Scrum-of-Scrums (SoS)	Jeff Sutherland and Ken Schwaber	Scrum Inc.	2001	Mechanism	Program	◑
Large-Scale Scrum (LeSS)	Craig Larman and Bas Vodde	LeSS Company B.V.	2008	Framework	Enterprise	●
Gill Framework (Gill)	Asif Qumer and Brian Henderson-Sellers	Adapt Inn	2008	Framework	Enterprise	○
Enterprise Transition Framework (ETF)	-	agile42	2011	Framework	Enterprise	◑
Mega Framework (Mega)	Rafael Maranzato, Marden Neubert, and Paula Heculano	Universo Online S.A.	2011	Framework	Portfolio	○
Scaled Agile Framework (SAFe)	Dean Leffingwell	Scaled Agile Inc.	2011	Framework	Enterprise	●
Disciplined Agile Delivery (DAD)	Scott Ambler	Disciplined Agile Consortium	2012	Framework	Enterprise	●
Enterprise Agile Delivery and Agile Governance Practice (EADAGP)	Erik Marks	AgilePath	2012	Set of practices	Enterprise	◔
Spotify Model (Spotify)	Henrik Kniberg, Anders Ivarsson, and Joakim Sundén	Spotify	2012	Model	Enterprise	◔
Recipes for Agile Governance in the Enterprise (RAGE)	Kevin Thompson	Cprime	2013	Framework	Portfolio	◔
Continuous Agile Framework (CAF)	Andy Singleton	Maxos LLC.	2014	Framework	Program	◔
Enterprise Scrum (eScrum)	Mike Beedle†	Enterprise Scrum Inc.	2014	Framework	Enterprise	◑
eXponential Simple Continuous Autonomous Learning Ecosystem (XSCALE)	Peter Merel	Xscale Alliance	2014	Set of principles	Enterprise	◑
Holistic Software Development (HSD)	Mike MacDonagh and Steve Handy	Holistic Software Consulting Ltd.	2014	Framework	Enterprise	◔
ScALeD Agile Lean Development (SALD)	Peter Beck, Markus Gärtner, Christoph Mathis, Stefan Roock, and Andreas Schliep	-	2014	Set of principles	Enterprise	◔
FAST Agile (FAST)	Ron Quartel	Cron Technologies	2015	Set of methods	Program	◔
Lean Enterprise Agile Framework (LEAF)	-	LeanPitch Technologies	2015	Framework	Enterprise	◑
Nexus (Nexus)	Ken Schwaber	Scrum.org	2015	Framework	Program	◑
Parallel Agile (Parallel)	Doug Rosenberg, Brarry Boehm; Matt Stephens, Charles Suscheck, Shobha Dhalipathi, and Bo Wang	Parallel Agile Inc.	2016	Set of methods	Enterprise	◔
Scrum at Scale (S@S)	Jeff Sutherland and Alex Brown	Scrum Inc.	2018	Framework	Enterprise	◑

To better understand the dynamics of the scaling frameworks movement, we recorded the version history of the participating frameworks and their inter-dependencies and plotted them in an evolution map (see P3). While the first framework, namely Crystal Family, was created in 1997, the real big breakthrough started after 2010, culminating in recent years (see P3). By comparing our results with Abrahamsson et al. [AWSR03], we observed two notable parallels between the movement of conventional agile methods and scaling frameworks. First, both movements emerged from parallel innovation by consultants aiming to support organizations in adopting or scaling agile methods. Second, similar to agile methods, scaling frameworks have continuously emerged and evolved since the movement started. This trend is expected to continue as the methodologists seemed to be committed to improving their frameworks in the future (see P3).

As part of the survey, we asked the creators of the frameworks what their intentions were in creating those. We identified 12 reasons grouped into four categories: complexity, customer, market, and organization (see P3). Most of the reasons fall into either the category of improving the organization's current state or dealing with the organization's prevalent challenges, which look similar to causes that trigger organizational changes [Sca21]. The most frequently cited reasons were to improve the agility/adaptability of organizations and to improve collaboration/coordination/synchronization among agile teams working on the same product (see P3). While several reasons, e.g., dealing with increased complexity or scaling agile practices to more people, were already described in previous studies (cf. [Paa17, PHK17]), we identified two reasons not reported by the existing literature: descaling large product organizations into smaller independent entities and improving customer involvement (see P3).

The survey also revealed 30 benefits of adopting scaling frameworks grouped into two categories: business/product and organization/culture (see P3). The most commonly mentioned benefits of adopting these frameworks were enabling frequent product deliveries, enhancing employee satisfaction/motivation/engagement, and providing customer/business value. The majority of the claimed benefits, e.g., enabling shorter feedback cycles or improving customer satisfaction, were similar to benefits associated with the adoption of conventional agile methods and already cited in previous studies (cf. [Dig20]). However, the creators did not mention general benefits of adopting agile practices, e.g., improved productivity [Dig20]. We extended the literature by two additional benefits, i.e., reducing headcount and fostering servant-leadership (see P3).

The survey resulted in the identification of 22 challenges grouped into three categories: implementation, organization/culture, and scope (see P3). The two most commonly mentioned threats associated with the adoption of scaling frameworks were: using scaling frameworks as cooking recipes and not understanding the reasons why they should be applied, which at the same time have also not yet been identified in the existing literature (see P3). Most of the identified challenges, e.g., change resistance, moving away from agile, implementation is difficult due to remaining power structures, and lack of management buy-in, were already reported in previous studies (cf. [DPL16, KHR18, CC19b]). Since the usage of these frameworks is not a silver bullet for scaling agile methods in large organizations or projects [PPL19], several creators pointed out that leaders and change agents should focus on changing the culture and mindset of the people instead of using these frameworks as cooking recipes (see P3).

An overview of the top-5 most cited reasons, benefits, and challenges is presented in Table 4.4.

Table 4.4. Reasons, benefits, and challenges of using scaling agile frameworks (based on P3)

Top-5 reasons	Top-5 benefits	Top-5 challenges
• Improving the agility/adaptability of the organization • Improving the collaboration/coordination/synchronization of agile teams working on the same product • Dealing with increased complexity • Descaling large product organizations in smaller independent entities • Enabling the information/communication flow between agile teams	• Enhancing employee satisfaction/motivation/engagement • Enabling frequent product deliveries • Improving software quality • Providing customer/business value • Fostering the creation of autonomous teams	• Using frameworks as cooking recipes • Using frameworks without understanding for what reasons they should be applied • Lack of management buy-in • Moving back from agile to traditional management approaches • Implementing is difficult due to framework complexity

Most case studies on scaling frameworks investigate the adoption of SAFe (cf. [PHK17, Paa17, Gus18]). The empirical analysis of other frameworks, such as LeSS, has mainly been disregarded, so we searched for potentially relevant cases to provide insightful findings regarding the adoption of LeSS. Finally, we identified a German automobile manufacturer as a valuable case study partner because it provided us with four different LeSS adoptions to study (see P4). While the case company's IT department had historically focused primarily on standardizing and cost-optimizing to run its IT, its management decided to transform the department into an agile product organization to improve its performance and agility. As part of the transformation, the department committed to converting all ongoing projects to agile projects and fully aligning the department with agile principles (see P4). Based on the success stories of their autonomous driving department, many IT projects decided to implement LeSS. We selected four of these projects for further investigation. While the four products comprised 3 to 15 agile teams, or 40 to 600 persons, mainly distributed over multiple locations, three adopted the basic LeSS framework, and one of them the LeSS Huge framework (see Section 2.3.2.2). As part of the investigation, we were primarily interested in (i) the reasons for adopting LeSS, (ii) the adaptations of the framework to fit their organizational context, (iii) the challenges they faced during its adoption, and (iv) the lessons learned they derived from its adoption (see P4).

The main reasons for adopting LeSS were handling the coordination between the teams and compensating for the shortcomings of adopting Scrum in large-scale projects (see P4). Additional reasons included LeSS's moderate degree of complexity with its "*more with LeSS*" tenet (see Section 2.3.2.2) while maintaining sufficient guidance for coordinating multiple agile teams working on the same product. The products selected LeSS to transform from traditional project thinking to product thinking and customer orientation (see P4). These reasons were also cited by the inventors of scaling frameworks in P3 and identified in previous studies (cf. [TSD19, EWC21]).

Confirming previous studies that companies typically tailor scaling frameworks to their structures and supplement these with additional practices to close their shortcomings (cf. [EP17, RRN+18, SB19]), the studied products also made some adjustments (see P4). For instance, they combined LeSS with existing lean and agile methods, e.g., Kanban, to manage features and track their progress using Kanban boards. They extended the standard LeSS framework to include a higher domain level to coordinate all products on a higher organizational level and align them with the organization's strategic objectives (see P4).

Although LeSS is minimalistic regarding providing additional guidance compared with other scaling frameworks [KHR18], the observed products had concerns regarding additional inter-team coordination meetings. Due to the shift of responsibilities of the middle management, managers feared they would be obsolete in the future (see P4). These challenges are common in scaling agile methods and were reported in previous studies (cf. [KHR18, TFW18]).

The studied products showed that clear communication of the overall picture could make the LeSS adoption smoother by obtaining the commitment of all stakeholders and allaying their fears. They reported that supplementary available training courses could facilitate the adoption of LeSS (see P4). In line with our findings, Dikert et al. [DPL16] also noted that communicating the goals of the transformation can help to remove confusion and make people understand the purpose of the change. Similarly, Dikert et al. [DPL16] identified the provision of additional training courses as a success factor in helping people become more positively inclined towards the new way of working and making them enthusiastic about change.

Table 4.5 summarizes the key findings of the embedded publications related to RQ2.

Table 4.5. Overview of key findings related to RQ2

RQ	No.	Key findings

RQ2	P2	• Consultants have initiated a large movement of scaling agile frameworks
		  • Identification of 22 scaling frameworks which are continuously emerging and evolving
		  • SAFe, LeSS, and DAD are one of the most widely adopted as well as mature frameworks
		• Identification of twelve key reasons behind the creation of scaling agile frameworks
		  • The creators mainly intend with creating scaling frameworks to improve the agility/adaptability of organizations and the collaboration of agile teams working on the same product
		• Identification of 30 benefits and 22 challenges of adopting scaling agile frameworks
		  • Enhancing employee satisfaction/motivation/engagement and enabling frequent product deliveries are the most cited benefits of adopting scaling frameworks
		  • The most indicated challenges associated with adopting scaling frameworks are using them as cooking recipes and not understanding the reasons why they should be applied
		• The adoption of LeSS promises several benefits but should be performed with caution
		  • With the adoption of LeSS, the German automotive manufacturer's products aim to enable the transformation from temporary projects to long-lasting products
		  • Standard practices of LeSS are extended to tailor it to the organization and close its shortcomings
		  • LeSS adoption may result in having dissatisfied managers retain their status and powers, which is why clear communication of the big picture of the adoption is crucial for a successful adoption

	P3	
	P4	

Key findings related to RQ3. Although early advice is that scaling agile methods is "*probably the last thing anyone would want to do*" [RME03], companies still try to do this [DFP19], which is why several face new challenges, e.g., dealing with a general resistance to changes or coordinating several large-scale agile activities [Ket07, DPL16]. Compared to the rich body of agile software development literature describing challenges (cf. [HBP09, ISM+15]) and best practices (cf. [BDS+99, CH04, BCS+10]), literature on large-scale agile development neglects to report challenges stakeholders face and adequate solutions to address them [UKCM18, UHM19]. Spurred by this research gap, we phrased the third research question as follows:

Research question 3 (RQ3)

What are the concerns of stakeholders in large-scale agile development, and how can they be addressed?

To answer the third research question, we conducted a structured literature review to identify typical challenges stakeholders face in large-scale agile development (see P5). We created a conceptual model to document best practices following the PDR method (see P6). Using the proposed conceptual model, we identified typical challenges of agile coaches and scrum masters and documented patterns to assist them (see P7).

In previous studies, several scholars have already endeavored to identify and describe the challenges of adopting agile methods at scale. For instance, Dikert et al. [DPL16] conducted a systematic literature review and identified 35 challenges, e.g., general resistance to change, lack of coaching, or internal silos kept, grouped into ten categories, e.g., agile difficult to implement or lack of investment. Another example is Shameem et al. [SCKK18], who performed a systematic literature review to reveal challenges related to the scaling of agile methodologies, e.g., lack of communication and lack of commitment. Since the problems identified were not directly linked to stakeholders, we attempted in P5 to obtain a complete overview of typical challenges of stakeholders. Using a structured literature review, we identified 79 challenges (see P5). While we identified 38 challenges that were reported in agile projects, e.g., managing technical debts or establishing self-organization, we uncovered 41 challenges that were novel, e.g., sharing a shared vision between the agile teams or managing dependencies to other existing environments (see P5). This finding confirms the early advice from the agile community stating that scaling agile methods is "*probably the last thing anyone would want to do*" [RME03]. Organizations not only struggle with challenges arising from adopting agile methods at the team level but also with issues emerging from scaling agile methods to higher organizational levels. The identified challenges were later grouped into 11 categories, as shown in Figure 4.5.

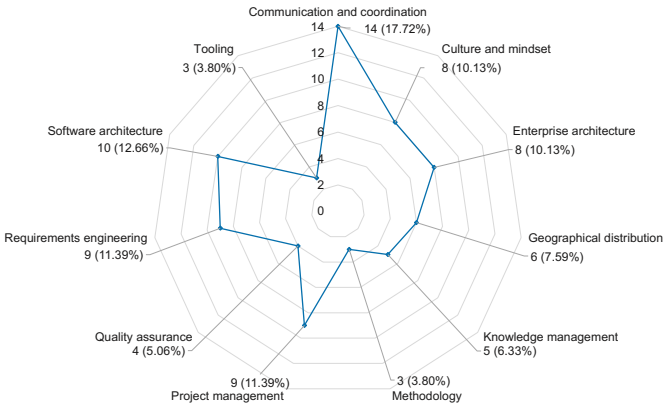


Figure 4.5.: Number challenges per category (based on P5)

We recorded many challenges regarding communication and coordination (see P5). This is not surprising since most companies have difficulties coordinating multiple agile teams. Thus, a large portion of the research conducted strives to understand and describe these issues (see P1).

Consultants developed various scaling frameworks to assist organizations in solving coordination issues (see P3). Comparing our results findings with those of Dikert et al. [DPL16], Shameem et al. [SCKK18], and Kalenda et al. [KHR18], we made interesting observations. All studies highlighted geographical distribution challenges. According to Shameem et al. [SCKK18], communication is a significant factor that may negatively influence large-scale agile development endeavors in geographically distributed environments. Similarly, Kalenda et al. [KHR18] stated that having development teams spread across several sites may experience significant challenges with agile development, as agile development emphasizes constant communication and team spirit. A distributed environment makes these close relationships hard to obtain. According to Dikert et al. [DPL16], coordination problems may occur when scaling agile methods over many geographically dispersed sites. The distribution of the teams may have adverse effects, e.g., reduced feelings of proximity when telecommunication is necessary or difficulties in arranging frequent meetings due to time zone differences. Moreover, Dikert et al. [DPL16] reported the problem of coordinating several agile teams' work as a central challenge. We had similar findings, as we identified six challenges related to geographical distribution, e.g., the working hours of cross-shore agile teams having to be synchronized or that companies have to deal with lacking team cohesion at different locations (see P5). Two studies highlighted challenges related to resistance to change and attachment to previous processes. While Kalenda et al. [KHR18] argued that the opposition could occur at all organizational levels, including development teams and middle and upper management, Dikert et al. [DPL16] stated that people are unwilling to change unless the change is perceived easy enough. Similarly, we revealed that people have doubts about changes, and obtaining buy-in, especially from the upper management, is essential (see P5). However, none of the studies mentioned any architectural challenges that accounted for about 20% of our identified challenges, e.g., integration issues and dependencies with other subsystems and teams, and finding the right balance between architectural improvements and business value (see P5). This observation is also in line with the general criticism that agile development and particular large-scale development have received as both lack focus on architecture [DM14, RWN+15]. On top of the previous studies, we could directly link the challenges to stakeholders facing them. For instance, while creating a proper upfront architecture design of the system might be complicated for software architects, agile coaches might have the duties to deal with incorrect practices of agile development. We created a list of stakeholders typically involved in large-scale agile development endeavors to do this mapping. We identified 14 stakeholders, e.g., agile coaches, enterprise architects, and scrum masters (see P5).

Table 4.6 shows the top-5 most reported challenges, challenge categories, and stakeholders.

We followed the recommendations by Meszaros and Doble [MD97] and analyzed several related pattern languages to get an idea of the state of the art regarding the documentation of best practices in large-scale agile development. We identified several pattern languages related to agile software development that practitioners and researchers have proposed (see P6). For instance, Coplien and Harrison [CH04] suggested more than 90 organizational patterns identified in successful agile projects and presented them in a concise, written form. These patterns can be applied to create new organizations from scratch or used to already existing organizations [CH04]. Another example is provided by Beedle et al. [BCS+10], compiling a collection of essential best practices for Scrum consisting of 11 patterns. The proposed pattern language can be applied by agile teams that want to start implementing Scrum. It also provides a way to

Table 4.6. Challenges and stakeholders in large-scale agile development (based on P5)

Top-5 challenges	Top-5 challenge categories	Top-5 stakeholders
• Coordinating multiple agile teams that work on the same product • Considering integration issues and dependencies with other subsystems and teams • Coordinating geographically distributed agile teams • Facilitating shared context and knowledge • Managing technical debts	• Communication and coordination • Software architecture • Project management • Requirements engineering • Culture and mindset	• Development team • Product owner • Scrum master • Software architect • Test team

organize the overall body of literature on Scrum, serving as a key entry point for beginners and experts looking for more knowledge about Scrum and related disciplines [BCS+10]. After the analysis of the pattern languages, we concluded the following shortcomings (see P6): (i) only a small fraction of the more than 500 identified patterns may be applicable to large-scale agile development (cf. [Scr21]), (ii) none of the proposed pattern languages categorized patterns in the way they should be executed, and (iii) none of them included the concept of stakeholders which practitioners highly desire (see P6). To address these deficiencies, in P6, we (i) created a conceptual model for documenting patterns in large-scale agile development, (ii) developed a pattern language, consisting of multiple patterns and concepts, and (iii) exemplified four patterns that can be applied to address concerns in large-scale agile development.

Inspired by the pattern language for EAM proposed by Ernst [Ern10] and further developed by Schneider and Matthes [SM15], we adopted the idea of creating a pattern language, including categorized patterns and stakeholders, and developed a conceptual model for documenting best practices in large-scale agile development. The proposed conceptual model includes three pattern types and four additional concepts (see P6):

- *Stakeholders* are persons who are actively involved in, have an interest in, or are in some way affected by large-scale agile development endeavors.
- *Concerns* can manifest themselves in many forms, e.g., goals, responsibilities, challenges, or risks.
- *Principles* are enduring and represent general guidelines that address given concerns by providing a common direction for action.
- *Coordination patterns* define proven coordination mechanisms to address recurring coordination concerns, i.e., managing dependencies between activities, tasks, or resources.
- *Methodology patterns* define concrete actions to be taken to address given concerns.
- *Viewpoint patterns* define proven ways to visualize information in form of documents, boards, metrics, models, and reports to address recurring concerns.
- *Anti-patterns* describe typical pitfalls and present revised solutions, which help pattern users to prevent these mistakes.

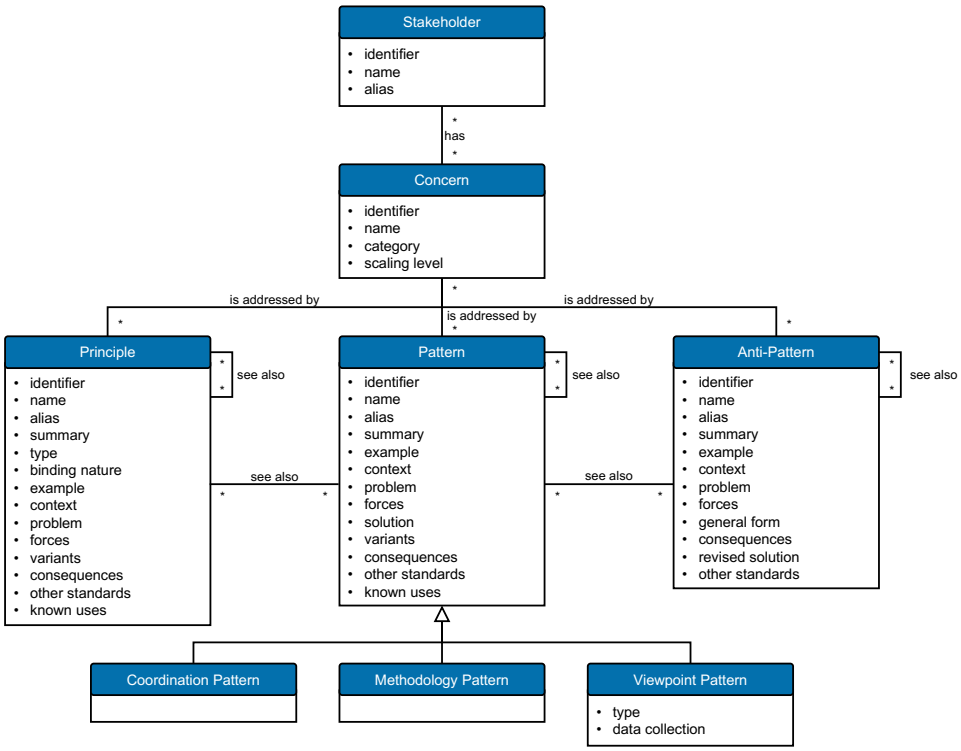


Figure 4.6.: Conceptual model for documenting patterns (based on P6)

According to Buschmann et al. [BHD07] and Fowler [Fow06], there is no ideal pattern form, and selecting a pattern form is a personal decision. It should also consider one's writing style and the ideas to be conveyed. Hence, we followed a similar template to document patterns in large-scale agile development to Buschmann et al. [BMR+96] and Ernst [Ern10] (see P6). Figure 4.6 depicts the conceptual model, including the proposed patterns types and concepts and the attributes used to document them.

To assess the structure and practical relevance of the conceptual model, we interviewed 14 experts from ten companies. In general, the proposed pattern types and concepts were found very valuable. Most respondents indicated that they would use patterns to address their concerns, helping them in their daily work (see P6). While the idea of patterns was met with positive feedback, several experts pointed out some risks. First, patterns are context-specific and may work very well in some organizations but not in others. Second, employees may focus too much on applying a pattern correctly rather than understanding the problem and intentions behind the pattern, similar to the over-reliance on scaling frameworks (see P3). We believe that the former risk can be mitigated by applying the PDR method, in which researchers could help companies select appropriate patterns and help them configure the patterns for their context. The second risk can be mitigated by using patterns to guide decision-making and not prejudging decisions. Some interviewees suggested using patterns for pedagogical purposes (see P6 and P7).

Based on observations and interviews as part of our research, we developed an initial version of a pattern language. We visualized it in a pattern graph within a prototypical web application

called "*Scaling Agile Hub*"[2]. Figure 4.7 provides the current version of the pattern language[3], which shows the different pattern types and concepts as nodes as well as their inter-relationships as edges. The four emphasized nodes in Figure 4.7 were exemplified in P6 (see Table 4.7).

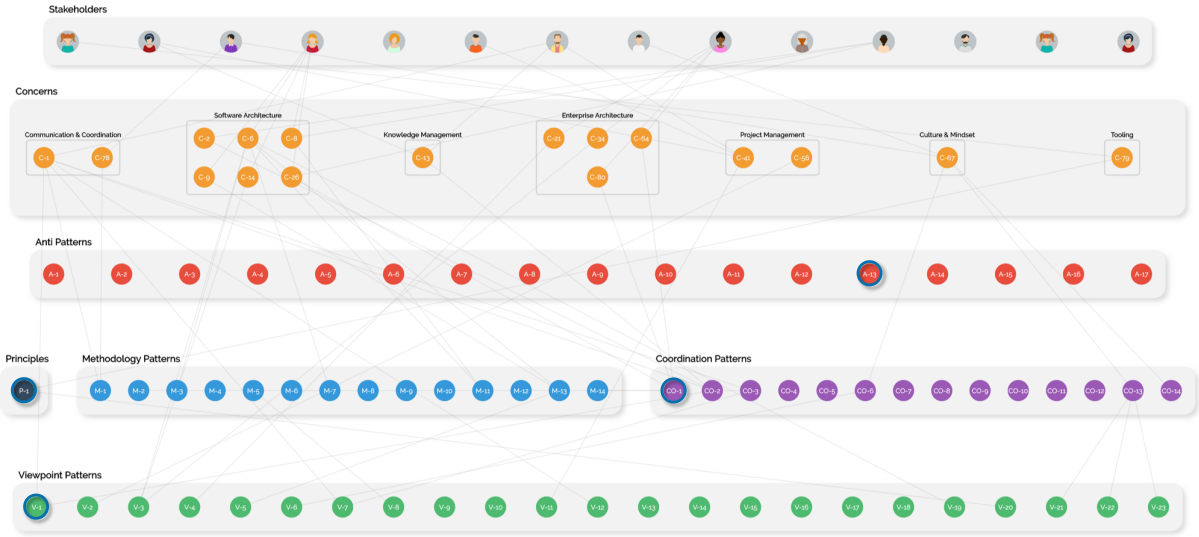


Figure 4.7.: Overview of the large-scale agile development pattern language (based on P6)

While the presented *Strictly Separate Build and Run Stages* principle aims to assist companies in releasing stable and reliable cloud-native platforms in large-scale agile development, the exemplified *Community of Practice* coordination pattern describes how organizations can build a platform for knowledge sharing about a specific domain, e.g., agile coaching, front-end architecture, information security. The *Iteration Dependency Matrix* viewpoint pattern provides a visual board that organizations can use to display the dependencies of agile teams working on the same product for upcoming iterations. The *Don't Use Agile as a Golden Hammer* anti-pattern points out the typical pitfall that companies obsessively tend to apply agile methods to all types of software projects, even when traditional methods would suffice, i.e., for simple projects with known technological implementation and requirements (see P6).

Since agile coaches and scrum masters are key facilitators of large-scale agile development endeavors [UM19] and are confronted with major difficulties in establishing an agile mindset across organizations [DPL16], we used in P7 the list of challenges compiled in P5 and the conceptual model proposed in P6. We interviewed 13 agile coaches and scrum masters to learn about their challenges and to reveal best practices they can use to address their problems.

Out of the 79 challenges revealed based on the literature in P5, 36 of them were mentioned by the interviewees, and additional 21 challenges were newly cited by the agile coaches and scrum masters, resulting in the identification of 57 challenges. While previous challenges, e.g., forming and managing autonomous teams and dealing with internal silos, were confirmed by the

[2] `https://scaling-agile-hub.sebis.in.tum.de/#/patterns`

[3] We conducted more than 100 semi-structured interviews with various roles from more than 20 industry partners. As a result, we identified 60 different patterns, which we plan to publish in a newer version of the presented pattern language and a pattern catalog.

Table 4.7. Overview of large-scale agile development patterns (based on P6)

Pattern name	Pattern summary	Pattern type	Addressed challenge(s)	Known uses
Strictly Separate Build and Run Stages	*Separate Build and Run Stages* ensures that an application's deployment phases are clearly separated.	Principle	• How can a cloud-native application be developed in a stable and timely manner?	5
Community of Practice	A *Community of Practice* are groups of people who share a concern, a set of problems, or a passion about a topic, and who deepen their knowledge and expertise in this area by interacting on an ongoing basis.	Coordination pattern	• How to create a platform for active knowledge sharing and discussion?	5
Iteration Dependency Matrix	*Iteration Dependency Matrix* visualizes dependencies among teams for the upcoming iterations.	Viewpoint pattern	• How to visualize dependencies between agile teams? • How to coordinate multiple agile teams that work on the same product? • How to consider integration issues and dependencies with other subsystems and teams?	4
Don't Use Agile as a Golden Hammer	*Don't Use Agile as a Golden Hammer* shows why it is not advisable to use agile methods in order to solve many kinds of problems.	Anti-pattern	• How to choose the correct software development approach? • How to decide whether agile methods should be used for a given project?	–

interviewees, they cited demonstrating the value of add of agile methods and the coaching of the higher management as significant challenges (see P7).

Based on the list of challenges, the interviewees were asked to outline practices to address them. We revealed 76 pattern candidates and 15 patterns after applying the "*rule of three*" [Cop96]:

- 21 Methodology pattern candidates $\Longrightarrow$ 5 Methodology patterns, e.g., *Piloting* and *Objectives and Key Results*.
- 18 Viewpoint pattern candidates $\Longrightarrow$ 2 Viewpoint patterns, i.e., *Global Impediment Board* and *Good Practices Newsletter*.
- 14 Coordination pattern candidates $\Longrightarrow$ 2 Coordination patterns, i.e., *Supervision* and *Community of Practice*.
- 12 Principle candidates $\Longrightarrow$ 4 Principles, e.g., *Celebrate Every Success* and *Consensus-Based Decisions*.
- 10 Anti-pattern candidates $\Longrightarrow$ 2 Anti-patterns, i.e., *Don't Use Frameworks as Recipes* and *Don't Overshoot Coordination Meetings*.

Similar to P6, we showcased five patterns in P7 that are summarized in Table 4.8.

The presented *Publish Good Practices* principle strives to assist companies in communicating experienced success stories and fostering a culture of continuous improvement. The showcased *Supervision* coordination pattern describes how organizations can build a platform for discussing

Table 4.8. Overview of patterns for agile coaches and scrum masters (based on P7)

Pattern name	Pattern summary	Pattern type	Addressed challenge(s)	Known uses
Publish Good Practices	*Publish Good Practices* to enable a culture of open communication and continuous improvement. By applying *Publish Good Practices*, agile teams are encouraged to talk about things that what went well and to share their achievements with other agile teams	Principle	• How to deal with doubts in people about changes? • How to facilitate shared context and knowledge? • How to build trust of stakeholders in agile practices? • How to establish a culture of continuous improvement? • How to establish a common understanding of agile thinking and practices?	5
Supervision	A *Supervision* offers agile teams a platform to discuss their current problems in a small and closed circle of participants and jointly find and evaluate solutions to these problems.	Coordination pattern	• How to encourage development teams to talk about tasks and impediments?	3
Global Impediment Process	The *Global Impediment Process* describes a structured process for identifying, documenting, and solving impediments that affect multiple agile teams.	Methodology pattern	• How to establish a culture of continuous improvement? • How to encourage development teams to talk about tasks and impediments?	5
Global Impediment Board	Global Impediment Board shows all impediments of an organization which are ether not solvable by an agile team itself or are relevant for several teams in a company.	Viewpoint pattern	• How to encourage development teams to talk about tasks and impediments?	5
Don't Use Scaling Agile Frameworks as a Recipe	*Don't Use Scaling Agile Frameworks as a Recipe* shows why it is not advisable to over rely on scaling agile frameworks without training people in agile values and principles and tailoring these frameworks to the specific needs of the organization.	Anti-pattern	• How to deal with incorrect practices of agile development?	–

current problems and solutions in small and closed circles of participants. The *Global Impediment Process* methodology pattern describes a structured process that organizations can use to identify, document, solve, or escalate impediments that affect multiple agile teams. The *Global Impediment Board* viewpoint pattern can be applied in combination with the *Global Impediment Process* methodology pattern. It provides organizations with a visual representation of a board that they can use to document global impediments in a uniform format and central manner accessible to the employees. The *Don't Use Scaling Agile Frameworks as a Recipe* anti-pattern hints at the typical fallacy of companies over-relying on agile frameworks without questioning whether they are helpful for addressing the actual problems of the company (see P6). The latter was of particular interest because most respondents used scaling frameworks to support their product developments. In the interviews, we noticed that the importance of scaling frameworks to companies varied widely. While some interviewees saw the proper implementation as critical, others viewed these frameworks as a means to an end and generally used them as inspiration (see P6). The former concentrated too much on properly implementing a framework and tended to use the frameworks as recipes (see *Don't Use Scaling Agile Frameworks as a Recipe* anti-pattern).

Although many creators suggest using their framework entirely and without modification, several interviewees mentioned that their companies did not adopt the frameworks one-to-one but adapted them to their needs (see P6). We observed two types of agile coaches during the interviews: process-oriented and mindset-oriented agile coaches. The process-oriented coaches mainly focused on the teams' proper application of agile practices and methods. They proposed best practices in the form of methodology patterns, e.g., *Objectives and Key Results* and *Global Impediment Process*. The mindset-oriented coaches often explained concerns about adopting an agile mindset across the company or creating an ideal working environment for agile teams. The mindset-oriented coaches typically named principles, workshops, and visualizations for resolving their concerns, e.g., *Publish Good Practices* principle and *Supervision* coordination pattern (see P6). Our observations align with Kelly's [Kel12] findings of two different coaching approaches: directive and non-directive coaching. While process-oriented coaches often use a directive approach, mindset-oriented coaches mostly use non-directive coaching. In directive coaching, agile coaches have extensive domain knowledge and train agile teams in the correct application of agile practices. In contrast, the non-directive approach does not require coaches to be domain experts. Instead, coaches seek to assist agile teams in focusing on their own goals and working towards achieving them. While non-directive coaching is more appropriate for teams already familiar with agile practices, the directive approach is more suited for new teams [Kel12].

Table 4.9 summarizes the key results of the embedded publications related to RQ3.

Table 4.9. Overview of key findings related to RQ3

RQ	No.	Key findings

RQ3	P5	• Identification of 79 challenges that stakeholders face in large-scale agile development
		  • The most cited challenges are coordinating multiple agile teams that work on the same product and considering integration issues and dependencies with other subsystems and teams
		  • Around 20% of the identified challenges are related to communication and coordination between agile teams as well as with other non-agile units

	P6	• Creation of a conceptual model for documenting pattern in large-scale agile development
		  • Using a combination of three pattern types and four concepts is perceived valuable by practitioners
		  • Using a pattern language and pattern graph can enable practitioners to quickly identify role-specific challenges and appropriate patterns

	P7	• Identification of 57 challenges, 76 pattern candidates, and 15 patterns for agile coaches and scrum masters
		  • A significant part of the challenges for agile coaches and scrum masters described in the literature is also encountered by them in practice
		  • The five showcased patterns can assist agile coaches and scrum masters in addressing challenges related to encouraging agile teams to talk about their impediments, establishing a culture of continuous improvement, and dealing with incorrect agile development practices

Key findings related to RQ4. The collaboration between agile teams and architects is crucial as "*agility is enabled by architecture, and vice versa*" [NÖK14, Wat14]. This topic has been disregarded in existing studies, as they do not provide sufficient recommendations on (i) how solution architects can be involved and (ii) how enterprise architects can support agile teams at higher levels. In this light, we formulated the fourth research question as follows:

Research question 4 (RQ4)

How do architects collaborate with agile teams and support them in large-scale agile development?

To address the fourth research question, we conducted a single-case study to investigate the collaboration between solution architects and agile teams on architectural issues in a large-scale agile development program of a German consumer electronics retailer (see P8). We conducted a multiple-case study to study the expectations of agile teams for enterprise architects and the responsibilities and challenges of enterprise architects (see P9 and P10).

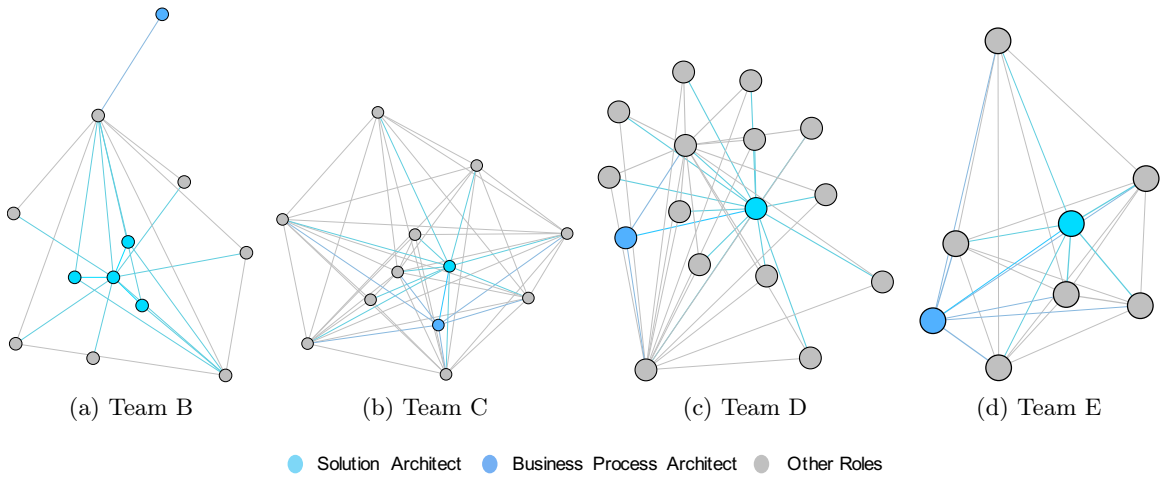
Previous studies have already indicated that the role of architects is demanding [DMFS18] and that large-scale agile development endeavors without any form of architectural guidance can hardly be successful [UHK+18]. Existing studies pointed out that architects face numerous challenges in these environments, e.g., poor communication with agile teams and a lack of understanding of the value of architecture complicating the advancement of architectural topics [Wat14, AMG16]. Although some studies presented tactics to incorporate architecture in large-scale agile development (cf. [BNO12, NÖK14]), they still lack a concrete description of how architects can be involved and how they communicate with agile teams. Against this backdrop, we searched for a case study partner to support us in answering the questions mentioned above. We identified a German consumer electronics retailer as a valuable case study partner. It provided us with a large-scale agile development program involving multiple architects and allowed us to perform social network analysis (see P8). In 2016, the German consumer electronics retailer decided to relaunch a failed customer relationship management project using agile methods. Due to the project's complexity, the company's management decided to relaunch the project using SAFe because of its proven track record in large enterprises and comprehensive documentation. The implementation of SAFe was initiated with a pilot project. After a few PIs, the responsible management team realized that SAFe did not provide sufficient guidance for coordinating agile teams, so they decided to combine SAFe with Spotify (see P8). At the time of observation, the program included 62 employees and consisted of eight teams, one responsible for coordination (Team A), four for actual development (Team B – Team E), and three belonging to the company's suppliers who provided external support for their third-party systems (Team F – Team H). In addition to typical SAFe roles, the program also included two architect roles, namely seven solution architects and four business process architects (see P8). As part of the study, we were mainly interested in (i) how the architects were integrated into the program and (ii) how they communicated with the agile teams (see P8).

Each team included 9 to 16 members from different areas of expertise, e.g., product owner, scrum master, developer, and solution and business process architect. The architects were actual members of the agile teams, which typically included one solution architect and one business process architect, except Team A with no architects and Team B with four solution architects (see P8). While each solution architect took care of the team's system architecture with its subsystems and interfaces, each business process architect was in charge of its team's business

architecture[4]. Both roles were responsible for making architectural decisions and guiding their teams to fulfill the required architectural standards (see P8).

To understand the communication of the architects with the teams, we explored how they exchanged architecture-related information both within the team and with other teams in the large-scale agile development program (see P8). To this end, we used social network analysis to visualize the information exchange and identify communication patterns (see Section 2.5).

Figure 4.8 visualizes the communication networks of each team, with team members represented as nodes and their architecture-related interactions represented as edges, which are shorter the more frequent they are[5]. An analysis of the intra-team communication networks in Figure 4.8 shows an *all-channel network* pattern in all teams, meaning that architecture-related interactions within the teams were decentralized and non-hierarchical, allowing all team members, regardless of their roles, to communicate freely and directly with all other members. This communication also coincides with the values and principles of agile development. The analysis also indicates that the solution and business process architects usually were located in the center of the communication networks, forming the leading role in architecture-related topics. They communicated with all team members, usually daily (except in Team B) (see P8). The comparison between solution and business process architects reveals that the former shared much more frequently in their teams about architecture topics than the latter. This finding is consistent with our observations that business process architects were newly introduced to the program, and their specific roles remained unclear (see P8). This finding is confirmed by calculating the normalized degree centrality C'_D for all solution and business process architects. While the average normalized degree centrality C'_D for all solution architects was 0.9775, the average normalized degree centrality C'_D for all business process architects was 0.4825 (see P8). An interesting observation could be made for Team B (see Figure 4.8a) as it was the only team having four solution architects. A centralized *wheel network* pattern was observed among the four solution architects, three of whom were externally and led by the internal solution architect (see P8).

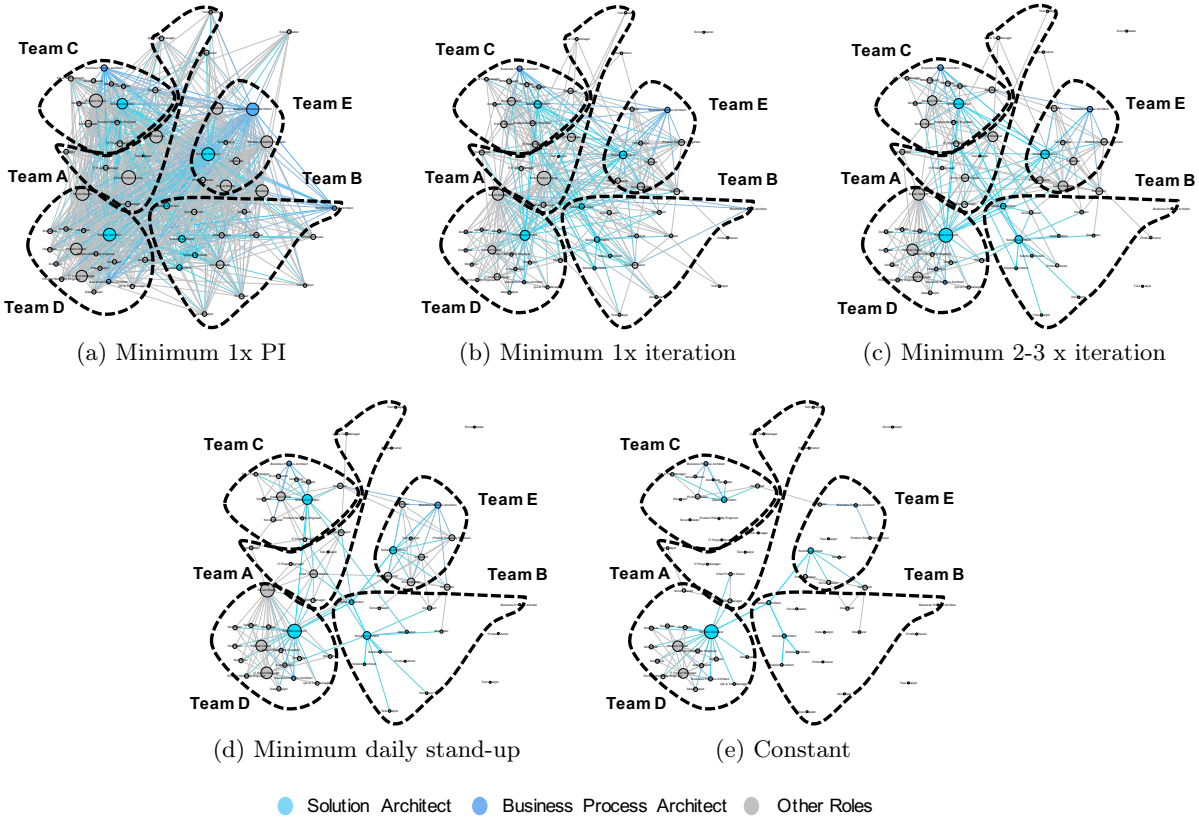


(a) Team B (b) Team C (c) Team D (d) Team E

Figure 4.8.: Intra-team architecture sharing (based on P8)

[4]At the time of observation, business process architects were new to the program.
[5]Team A is not included in this analysis because it had no architects.

Figure 4.9 displays the communication networks in the large-scale agile development program in chronological order. While Figure 4.9a indicates interactions that occur at least every PI (two and a half months), Figure 4.9e implies constant exchanges that occur several times a day. A general inspection of Figure 4.9 shows that Team A, which has the coordinating role for the program, is located in the center of the graphs, indicating that this team frequently exchanged information with all other teams related to architectural topics (see P8). Furthermore, a close collaboration was observed between Team B and Team E, and between Team B and Team D, due to architectural dependencies between the systems they worked on. Figure 4.9 also shows that the solution and business process architects were intensely involved in the architectural exchange, indicated by large nodes. This observation suggests that they were not only sharing information within their teams but also with others outside their teams, especially with other architects. The normalized degree centrality C'_D of the solution and business process architects shows that solution architects from Team D and Team E and the business process architect from Team E were very actively involved in the inter-team architecture exchange, as they are also represented as large nodes (see P8). A temporal analysis of Figure 4.9 shows that all individuals were involved in architecture-related interactions between the teams on an occasional basis (see Figure 4.9a). At the same time, the solution and business process architects constantly facilitated the exchange of architecture-related information between the agile teams (see Figure 4.9e).



(a) Minimum 1x PI (b) Minimum 1x iteration (c) Minimum 2-3 x iteration

(d) Minimum daily stand-up (e) Constant

Figure 4.9.: Inter-team architecture sharing (based on P8)

Although some articles are available that describe the role of enterprise architects (cf. [CCJ+18, HDSD20]), they still lack to provide a cross-case analysis of how enterprise architects should support large-scale agile development endeavors. We agree with Hanschke et al. [HEK15] and Canat et al. [CCJ+18] that more empirical research is needed in this context. Motivated by this backdrop, we sought out case study partners, having several ongoing large-scale agile development endeavors and well-established EAM functions, helping us to answer the above questions. We selected five cases from different industries that were based in Germany, i.e., a global insurance company (with 140,000+ employees), an automobile manufacturer (with 130,000+ employees), an IT company (with 7,000+ employees), a consumer electronics retailer (with 50,000+ employees), and a public sector insurance company (with 6,700+ employees) (see P9 and P10). We conducted 38 interviews with various roles, e.g., developers, enterprise architects, and managers. All case companies started enterprise-wide change processes between 2015 and 2017. The main reasons for these change processes were to create modern working environments for employees, improve the time-to-market of new products, and increase customer satisfaction, to name a few. Some change processes started with large pilot projects, others with large pilots in entire business units. All cases used a combination of different scaling frameworks to support their change processes, most notably SAFe, LeSS, Spotify, and a customized and scaled version of Scrum. As part of the study, we were primarily interested in (i) agile teams' expectations of enterprise architects and the extent to which these are met, (ii) enterprise architects' responsibilities and ways of working in supporting agile teams, (iii) the challenges of enterprise architects in supporting agile teams (see P9 and P10).

We used the organization-specific agile EAM practice model proposed by Hauder et al. [HRSM14] to identify the agile teams' expectations for enterprise architects and how these are met. We used the agile EAM model and focused on the activities, which are relevant for the collaboration that consist of seven aspects, namely (i) models, (ii) availability, (iii) communication, (iv) involvement, (v) support, (vi) feedback, and (vii) recommendation (see P9).

While enterprise architects provided a wide range of architectural models for agile teams, e.g., application landscape diagrams, business capability maps, and data models, the agile teams' primary expectations were that the models should be provided timely, include a description of their binding nature, have an appropriate level of quality and consistency, and reflect details relevant to the agile teams. Other mentioned expectations were that the models should provide clear added value for the agile teams and be applicable (see P9). The interviewees felt that the architecture models provisioned by the enterprise architects only partially met the agile teams' expectations. The agile teams generally considered the models too abstract or too unspecific and therefore judged them irrelevant. Some respondents also indicated that the models were not provided on time. Interestingly, enterprise architects perceived the models most negatively, acknowledging that the models did not provide sufficient guidance for implementation by agile teams and were therefore mainly considered impractical (see P9).

In all companies, the general statement from the interviewees was that enterprise architects should not be part of agile teams and should remain in their overarching role. Agile teams expected enterprise architects to be available when needed to support and consult them on architecture issues. However, primarily due to capacity constraints, this expectation was only partially met as agile teams reported having difficulty finding enterprise architects. The commu-

nication between agile teams and enterprise architects mainly occurred indirectly through third roles, e.g., solution and domain architects (see P9). Some agile teams were able to communicate directly with enterprise architects via phone, email, or face-to-face meetings. In some companies, enterprise architects and agile teams were co-located in the same building, improving their communication due to shorter physical and communication distances. In instances where enterprise architects were directly involved in large-scale agile development projects, the communication between enterprise architects and agile teams was mainly facilitated via agile events, e.g., daily scrums, sprint plannings, or sprint retrospectives (see P9).

In all cases, the agile teams expected more frequent and personal contact with the enterprise architects and considered communication via third parties sub-optimal (see P9). Several agile teams indicated that their expectations for communication with the enterprise architects were not met. They argued that enterprise architects had less time, lacked technological expertise, and did not provide appropriate communication channels, affecting the speed and quality of the communication. They also indicated that direct communication channels were lacking and that they had to put a lot of effort into finding the right persons. Similar observations were made by Canat et al. [CCJ+18], stating that the communication between enterprise architects and agile teams is a major issue. Some agile teams found communication with enterprise architects valuable through the existence of direct communication channels and open dialogues (see P9).

We noticed that the involvement of agile teams in relevant architectural processes differed across the cases and that agile teams expected to be involved in the creation of intentional architectures. Although agile teams follow the emergent design principle of the Agile Manifesto [BBvB+01], most of them felt left out regarding their involvement in the architecture process because of weak collaboration. They called for greater participation in the creation of an intentional architecture. Interestingly, they did not insist on full autonomy as long as they were involved in architectural decision-making (see P9). Several enterprise architects shared this view and stated that they could not involve agile teams due to their fully exhausted capacity. Some managers indicated that not all agile teams could be involved in architecture processes as this would constitute a scaling issue, and they lack an overarching view of the company (see P9).

Enterprise architects supported agile teams by providing architectural principles, tools, technology stacks, technical roadmaps, and architectural advice and support. In some cases, enterprise architects even experimented with new technologies and collaborated with agile teams on a code basis. Agile teams primarily expected enterprise architects to provide practical solutions and support in selecting new tools and personal support and guidance in realizing architectural target states (see P9). We agree with Drews et al. [DSHT17] and Kulak and Li [KL17] and believe that enterprise architects should also focus on technologies and keep their technological knowledge up-to-date to provide adequate support for agile teams. We noticed that most agile teams were not satisfied with the current circumstances. They complained about the lack of support from enterprise architects due to capacity problems and a lack of technological know-how. Interestingly, enterprise architects had similar perceptions and stated that they could not provide adequate support due to time restrictions and missing technological know-how. Several observed EAM initiatives tried to address these challenges by adopting agile and lean methods. This observation is consistent with Hauder et al. [HRSM14] and Hanschke et al. [HEK15], stating that the number of EAM initiatives using agile and lean practices to improve their efficiency

and productivity is increasing. Most managers had a positive perception and praised that their enterprise architects received positive feedback and were demanded by agile teams (see P9).

We observed that no formal feedback processes between agile teams and enterprise architects existed. Agile teams were only able to provide feedback informally and as needed. In this respect, agile teams could use typical communication media, e.g., e-mail, face-to-face meetings, or CoPs. Agile teams expected to have regular appointments with enterprise architects or continue to use the CoPs or personal dialogues for feedback purposes. In all case companies, the interviewees appreciated the enterprise architects' reflection on the feedback of agile teams. They stated that the enterprise architects reflected the input from agile teams, e.g., by providing new or revised architecture artifacts or attempting to work more closely with agile teams (see P9). Some agile teams stated that their expectations were not fulfilled due to the lack of a feedback culture and mechanisms. The majority of the enterprise architects felt that the expectations of the agile teams for the opportunity to provide feedback were met and explained this by listing some feedback opportunities, e.g., workshops, e-mail, or personal face-to-face meetings (see P9).

We used the Net Promoter Score (NPS) and asked the three stakeholder groups: agile teams, enterprise architects, and managers, the question: *How likely would you recommend an enterprise architect to an agile team?* Figure 4.10 shows the results of the NPS calculations for the three stakeholder groups (see P9). While all interviewed managers would recommend enterprise architects ($NPS = 100.00\%$), agile teams were not satisfied with the current support ($NPS = -37.50\%$). Interestingly, while most of the interviewed enterprise architects would recommend themselves and seemed satisfied with how they support agile teams, some were also critical of their current support ($NPS = 76.92\%$) (see P9). We deduced that managers had a positive perception and agile teams had a negative attitude towards enterprise architects. This observation indicates that the added value of enterprise architects has not reached the team level, which is a similar result to that of Canat et al. [CCJ+18].

Figure 4.11 provides an aggregated overview of each stakeholder group's assessment of how enterprise architects met agile teams' expectations alongside the seven investigated aspects. Our results reconfirm that the added value of enterprise architects has not reached agile teams. In addition, we observed that enterprise architects' self-perceptions and the external perceptions of others differed. Enterprise architects seemed confident about their support, agile teams had a rather negative perception and seemed less satisfied with the provided support (see P9).

We were interested in determining what responsibilities enterprise architects had and how they changed their working methods. We identified 17 responsibilities of enterprise architects (see P10). In addition to typical tasks, e.g., developing architectural artifacts like standards, roadmaps, and principles, enterprise architects were also expected to discover and test new technologies or identify and manage dependencies between agile teams (see P10). Enterprise architects were responsible for developing a shared understanding of architecture across the company, e.g., organizing training on architecture fundamentals or CoPs to facilitate sharing information and insights on architecture topics. With the ongoing change processes in the case companies, the architectural decision-making authority shifted from the enterprise architects to the agile teams. Enterprise architects were responsible for supporting the architectural decision-making process of the agile teams (see P10). Along with this change, enterprise architects worked with agile teams and argumentatively convinced them so that they worked toward the envisaged

Figure 4.10.: Net promoter score of enterprise architects (based on P9)

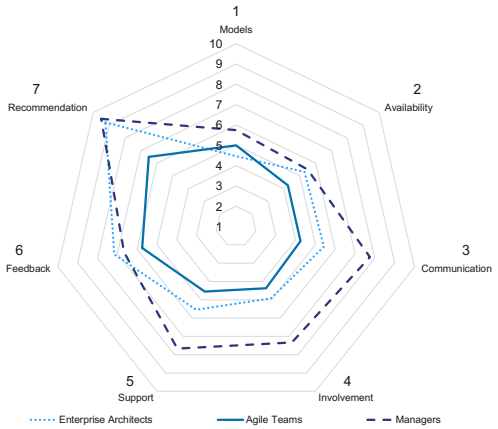


Figure 4.11.: Expectation fulfillment by enterprise architects (based on P9)

architectural direction. Enterprise architects were viewed as service providers and consultants responsible for providing tools, offering consultation for architectural issues, or assisting agile teams in resolving architectural impediments (see P10). Similar observations were made by Babar et al. [BBM13], stating that agile teams are increasingly empowered to make local architectural decisions. This change entails a paradigm shift of mindsets, i.e., enterprise architects take on the role of "*advisory servant*" rather than a "*command and control*" philosophy. Like Drews et al. [DSHT17], we witnessed that enterprise architects consulted agile teams to realize architectural standards and acted as facilitators for inter-team architecture exchange.

We observed several changes in the enterprise architects' working with the changing responsibilities. We identified 15 changes in the working methodology of enterprise architects (see P10). The most notable changes were regarding weaning the two typical anti-patterns of enterprise architects, namely "*Big Design Upfront*" and "*Ivory Tower*" [UM20b]. Instead of excessive efforts to create "*perfect*" and "*outdated*" architecture models that typically involve tedious conceptualization phases, enterprise architects adopted agile and lean practices to create evolving and simple architecture models. Instead of working in isolation and creating artifacts that do not address genuine concerns of agile teams, enterprise architects spent most of their time communicating and collaborating with agile teams. For instance, they enabled a collaborative process to create target architectures with agile teams (see P10).

The interviewees cited 17 challenges that enterprise architects faced (see P10). Across all companies, the interviewees cited the challenge of enterprise architects having to deal with the agile teams' lack of understanding of the value of enterprise architecture. Originating from this difficulty, agile teams tended to view external recommendations from enterprise architects with skepticism. As a result, enterprise architects without decision-making authority had problems driving the case companies' architecture strategies (see P10). The interviewees pointed out some challenges that arose from the tension between agile and architecture, similar to those reported by Hanschke et al. [HEK15]. For example, while agile teams were under pressure to deliver new features and had less time for architectural improvements, enterprise architects were responsible for building sustainable architectures that reflected companies' long-term strategies. Because of

this tension, the case companies reported the accumulation of new technical debts. This observation is consistent with the findings of Dingsøyr et al. [DMFS18], stating that the pressure of agile teams to deliver business functionality can lead to negligence of long-term architectural improvements. Other reported challenges emerged from the high frequency of changes made in the solutions, which required the enterprise architects to do more continuous architectural work (see P10). Enterprise architects' lack of capacity was also reported as a challenge, wherefore they had difficulties tracking and evaluating the progress of agile teams in achieving predefined architecture goals. The impact of the change processes on architectural decision-making led to confusion regarding the distribution of responsibilities for architectural issues (see P10).

Table 4.10 shows the top-5 most reported responsibilities, changes in the way of working, and challenges of enterprise architects.

Table 4.10. Duties, way of working, and challenges of enterprise architects (based on P10)

Top-5 responsibilities	Top-5 way of working	Top-5 challenges
• Collaborating with stakeholders to develop the company's architectural roadmap • Collaborating with agile teams to guide them through architectural requirements and roadmaps • Facilitating architectural decision-making process of agile teams • Creating and communicating architecture principles and ensuring their compliance • Organizing communities of practices and architecture boards	• Avoiding big design upfront by creating the simplest architecture that can possibly work • Leaving the 'ivory tower' by closely collaborating with agile teams • Involving agile teams in the architectural decision-making process instead of deciding over their heads • Using new tools such as collaboration tools of agile teams • Creating architecture roadmaps with shorter	• Dealing with a lacking understanding of enterprise architecture • Balancing short-term and long-term planning • Balancing upfront and emergent architecture • Dealing with acceptance issues by agile teams • Dealing with the loss of decision-making power

Table 4.11 summarizes the main results of the embedded publications related to RQ4.

Table 4.11. Overview of key findings related to RQ4

RQ	No.	Key findings

RQ4	P8	• Solution and business process architects are actively involved in large-scale agile development endeavors
		  • Both roles are team members of agile teams, taking care of the teams' system and business architectures
		  • Both intra-team and inter-team architectural exchanges are driven primarily by the two architect roles, allowing all members of the teams to communicate freely and directly with each other
		• Enterprise architects are expected to take both a coordinating and overarching role as well as an advisory and servant leadership role in supporting agile teams
		  • Agile teams have multifaceted expectations for enterprise architects, including providing up-to-date architectural models, having direct and frequent communication, providing technical guidance, and establishing an open feedback culture
		  • While managers have a very positive perception of enterprise architects and consider expectations to be met, agile teams have a rather negative perception
		  • Apart from traditional architectural duties, enterprise architects facilitate the architectural decision-making process of agile teams and organize communities of practices for architecture
		  • The daily work of enterprise architects shows a dramatic shift, with them adopting agile and lean practices and spending most of their time collaborating with agile teams on architectural matters
		  • Due to the lack of decision-making power, enterprise architects have difficulties driving architectural issues with agile teams

	P9	
	P10	

4.2. Implications for Research

The results of this dissertation have several implications for theory. Below, we summarize the theoretical contribution of the dissertation and its embedded publications, along with the four research questions (see Section 1.2) that guided the dissertation.

Implications for research related to RQ1. Leveraging the advantages of systematic mapping studies that provide a broad overview of research landscapes [PFMM08], our results help researchers to get a sound introduction to the current state of research by providing researchers (i) an overview of the most popular publication venues where the research community meets to discuss research findings, interests, and possible future research collaborations, (ii) a list of influential studies that researchers can use as foundational literature to build their research efforts upon, and (iii) an outline of the central research themes to more easily situate their research in the landscape and find researchers with similar research interests (see P1). Based on the analysis of various research facets, e.g., research types and research approaches, our findings comprise a classification of the current literature that researchers can use to identify and address research gaps, e.g., developing conceptual models and theories to strengthen the theoretical foundations of large-scale agile development research or creating rigorously developed frameworks, methods, and tools to assist practitioners (see P1). Our findings outline a research agenda consisting of a thorough description of research gaps alongside the identified open research questions that researchers can use as a starting point for their new research efforts (see P1).

Implications for research related to RQ2. While most studies either focus on a specific framework or provide a comparison of the most popular ones [EWC21], our results comprise a comparative list of 20 scaling frameworks whose real-world application can be explored by researchers (see P2). Unlike previous works, our results include a breakdown of the benefits and challenges for each scaling framework, which researchers can build upon to quantitatively assess and validate their inventors' claimed benefits and challenges, providing complete evidence on scaling frameworks that can be useful to practitioners (see P3). Because we conducted an exploratory single-case study due to the lack of detailed prior research, researchers can use our findings to build and strengthen theories about the application of scaling frameworks in industry and explain their adoption in organizations (see P4).

Implications for research related to RQ3. The large-scale adoption of agile methods entails various difficulties for companies [DPL16]. Hence, our findings comprise a list of 79 challenges that researchers can use to determine which of the listed challenges are occurring in practice and the most serious ones that must be addressed as they could harm large-scale agile endeavors (see P5). To address the observed challenges in practice, researchers can help organizations in selecting and adopting our proposed patterns and derive a set of lessons learned from their practical application, which researchers can, in turn, share with the research community (see P6 and P7). Researchers can also use our proposed conceptual model to document their own best practices, which can be adopted in organizations to address their issues (see P6).

Implications for research related to RQ4. Although there is rising interest from academia and industry to connect the two concepts of agility and architecture [BKNÖ14, RWN+15], empirical studies on exploring the collaboration between architects and agile teams are still scarce [HEK15, CCJ+18]. Thus, our results include a description and several communication

network graphs of how business process and solution architects collaborate with agile teams, which scholars can build upon to assess the effects of communication patterns on the performance of agile teams (see P8). As we performed an exploratory multiple-case study due to the lack of previous research, scholars can use our results to explain the role of enterprise architects in assisting large-scale agile development efforts and in developing new theories (see P9 and P10).

4.3. Implications for Practice

In addition to the implications for theory, this dissertation also has several implications for practice, providing practitioners with actionable insights into the emerging phenomenon of large-scale agile development. In what follows, we will discuss the practical implications based on the four research questions that guided the dissertation.

Implications for practice related to RQ1. The lack of shared understanding of the term "*large-scale agile development*" can lead to misunderstandings and conflicts among stakeholders involved in large-scale agile development endeavors, making it challenging to collaborate with others. Our findings enable practitioners to have a common terminology and perspective on large-scale agile development by describing how the existing literature characterizes this term based on reported case characteristics (see P1). Practitioners can use our results to identify (i) relevant venues where scientific knowledge is created to take inspiration for solving problems that researchers have already addressed and (ii) researchers who contribute to this area to catch future collaboration opportunities (see P1). We also present a logical organization of the research field utilizing a topic map that practitioners can use to effectively locate research results that can be used to address their problems related to large-scale agile development (see P1).

Implications for practice related to RQ2. Selecting a particular scaling framework is problematic for companies due to the lack of assessment models to guide critical decisions about adopting a specific framework [CC19b]. As a result, the selection is often not done systematically but merely based on a framework's popularity, the recommendations of consultants, or an ad hoc decision based on reading a book or attending a talk [DST18, CC19b]. Our results support practitioners to make informed decisions when selecting scaling frameworks by providing a comprehensive list of these frameworks and their key characteristics (see P2). Our findings furnish practitioners with the main factors they should consider when deciding on adopting scaling frameworks by providing a thorough description of the relative strengths, weaknesses, and challenges they might face when adopting these frameworks (see P3). Practitioners can use our findings as decision support by learning how a German automobile manufacturer adopted LeSS and what challenges and success factors were encountered during the adoption to ensure a successful implementation of a scaling framework in their companies (see P4).

Implications for practice related to RQ3. Although adopting agile practices at scale has proven to be very difficult, and companies are facing unprecedented challenges [DD08, DPL16], companies still try to adopt these methods in larger projects and organizations [DFP19]. Accordingly, our findings provide a single resource for practitioners, guiding them on the challenges they can expect to face when scaling agile methods, and a link to empirical research, providing further information about the challenges and potential approaches to resolving them (see P5).

Our findings include a comprehensive list of patterns and several detailed pattern descriptions, particularly related to agile coaches and scrum masters, that practitioners can use as a reference book to address their challenges (see P6 and P7)). Practitioners can use the proposed conceptual model as a template for the structured documentation of their proven solutions (see P6).

Implications for practice related to RQ4. In large-scale agile development, agile teams and architects need to work together, which does not always work smoothly due to the two stakeholder groups' antithetic ways of thinking and mindsets [KL17]. Practitioners looking for a suitable way to involve enterprise architects in their multi-team environments can use our findings to compare their current setups with the presented cases to learn how a fruitful collaboration between architects and agile teams can be achieved (see P7). Our findings help practitioners become aware of the responsibilities of enterprise architects and the expectations of agile teams so that they can be more tolerant of each other and work better together (see P7 and P8). Practitioners can use our results related to the enterprise architects' way of working to learn how their architects should act in a large-scale agile context and provide appropriate training to educate them in this direction (see P8).

CHAPTER 5

Limitations

This dissertation is subject to several limitations, which we discuss in the following, along with the four major characteristics that express the validity of a research study [Bha12, WRH+12, Yin15]: internal, external, construct, and conclusion validity. Table 5.1 summarizes the potential validity threats and the measures taken to counteract them.

Internal validity is concerned with factors affecting the relationship between the research process and the obtained results [RH09]. In the secondary studies, P1, P2, and P5, internal validity primarily focuses on the analysis of the extracted data [ESSD08], which a researcher's biased analysis may threaten. Multiple researchers performed all data analyses to mitigate this risk. The initial data analysis to create the solution artifacts in P6 and P7 may also be subjectively biased, which was remedied by multiple researchers performing the data analyses. Respondents' bias may threaten the internal validity of the survey in P3. We counteracted this threat by contacting the creators of the frameworks via emails and ensuring that the right persons answered the survey. This threat is irrelevant for P4, P8, P9, and P10 because all case studies are exploratory and refrain from theory building and making causal relationships.

External validity examines to what extent the obtained results can be generalized [RH09]. For the secondary studies, P1, P2, and P5, external validity refers to the representativeness of the selected papers regarding the overall goals of the secondary studies [ESSD08]. We followed comprehensive search processes to counteract this threat and designed the secondary studies to be as inclusive as possible. To mitigate risks related to the external validity of the survey in P3, we sought to collect responses from all existing scaling frameworks. We received responses from 15 creators and could not gather responses from seven creators, despite contacting them several times. Thus, this threat could not be eliminated. However, we did receive responses from the most widely adopted scaling frameworks, e.g., SAFe, LeSS, and DAD [Dig21]. The single-case studies in P4 and P8 focused on analytical generalization [RH09] by thoroughly describing the

Table 5.1. Summary of potential validity threats and primary countermeasures

RQ	No.	Internal validity	External validity	Construct validity	Conclusion validity
RQ1	P1	☒ Subjective biased data analysis ☑ Data analysis by multiple researchers	☒ Unrepresentativeness of selected studies ☑ Inclusive and comprehensive search process	☒ Missing relevant studies ☑ Usage of common electronic databases	☒ Incorrect data extraction ☑ Applying best practices and guidelines
RQ2	P2	☒ Subjective biased data analysis ☑ Data analysis by multiple researchers	☒ Unrepresentativeness of selected studies ☑ Inclusive and comprehensive search process	☒ Missing relevant studies ☑ Usage of common electronic databases	☒ Incorrect data extraction ☑ Applying best practices and guidelines
RQ2	P3	☒ Respondent bias ☑ Selective addressing of the methodologists	☒ Non-generalizability of the results ☑ Responses from most popular frameworks	☒ Inadequate questionnaire design ☑ Compilation of questionnaire based on previous studies	☒ Drawing incorrect conclusions ☑ Data interpretation by multiple researchers
RQ2	P4	• Not relevant as study is exploratory	☒ Non-generalizability of the results ☑ Analytical generalization	☒ Inaccurate description of the reality ☑ Triangulation of data sources and observers	☒ Drawing incorrect conclusions ☑ Creation of a case study database
RQ3	P5	☒ Subjective biased data analysis ☑ Data analysis by multiple researchers	☒ Unrepresentativeness of selected studies ☑ Inclusive and comprehensive search process	☒ Missing relevant studies ☑ Usage of common electronic databases	☒ Incorrect data extraction ☑ Applying best practices and guidelines
RQ3	P6	☒ Subjective biased data analysis ☑ Data analysis by multiple researchers	☒ Impracticality of the solution artifacts ☑ Feedback from practitioners	☒ Improper design of the solution artifacts ☑ Usage of existing documentation formats	☒ Inconsistency of the solution artifacts ☑ Creation of solution artifacts by multiple researchers
RQ3	P7	☒ Subjective biased data analysis ☑ Data analysis by multiple researchers	☒ Impracticality of the solution artifacts ☑ Feedback from practitioners	☒ Improper design of the solution artifacts ☑ Usage of existing documentation formats	☒ Inconsistency of the solution artifacts ☑ Creation of solution artifacts by multiple researchers
RQ4	P8	• Not relevant as study is exploratory	☒ Non-generalizability of the results ☑ Analytical generalization	☒ Inaccurate description of the reality ☑ Triangulation of data sources and observers	☒ Drawing incorrect conclusions ☑ Creation of a case study database
RQ4	P9	• Not relevant as study is exploratory	☒ Non-generalizability of the results ☑ Literal and analytical generalization	☒ Inaccurate description of the reality ☑ Triangulation of data sources and observers	☒ Drawing incorrect conclusions ☑ Creation of a case study database
RQ4	P10	• Not relevant as study is exploratory	☒ Non-generalizability of the results ☑ Literal and analytical generalization	☒ Inaccurate description of the reality ☑ Triangulation of data sources and observers	☒ Drawing incorrect conclusions ☑ Creation of a case study database

Legend:
☒: Limitation
☑: Countermeasure

cases to mitigate potential threats to external validity. The multiple-case studies in P9 and P10 focused on the cases' literal replication and analytical generalization. The external validity of the created solution artifacts in P6 and P7 may be threatened by the impracticality of the proposed pattern language, patterns, and concepts. To mitigate this risk, we interviewed 14 experts from ten companies to assess the practical relevance of the proposed pattern language. After creating the solution artifacts, the experts were contacted again and asked for feedback on the designed patterns and concepts.

Construct validity reflects what degree operational measures are studied represent what the researcher has in mind [RH09]. The secondary studies, P1, P2, and P5, may suffer from the potential incompleteness of the search results [ESSD08]. To mitigate construct validity threats, we searched in the common electronic databases to identify relevant studies, e.g., ACM Digital Library and IEEE Xplore, being the most prominent scientific databases in the software engineering field [KB13, PVK15], Science Direct, Web of Science, and Scopus, providing broad coverage of diverse research fields [RHL+17], and AIS eLibrary, containing articles from the primary information systems research outlets [Oat11]. The construct validity of the survey in P3 refers to whether the questionnaire's questions represent the attributes being measured. Thus, the inadequate design of the questionnaire may pose a significant risk to construct validity. Hence, the questions were compiled based on previously published studies. In case study research, construct validity concerns how well the description of the case represents reality [Yin15]. The construct validity of the case studies conducted in P4, P8, P9, and P10 may be threatened by an inaccurate description of the observed phenomena. As suggested by Stake [Sta95a], we improved the construct validity of the case studies by triangulating the data sources, i.e., collecting data by several methods and interviewing various persons and roles, and triangulating the investigators, i.e., involving multiple researchers in the data collection process. The construct validity of the solution artifacts in P6 and P7 may be threatened by an improper design. The proposed pattern and concept types were inspired by Ernst [Ern10], and the documentation of the patterns and concepts followed a template similar to Buschmann et al. [BMR+96] and Ernst [Ern10] to improve their construct validity.

Conclusion validity concerns to what extent the data and the analysis are dependent on the specific researcher [RH09]. In the case of the secondary studies, P1, P2, and P5, this threat refers to factors such as incorrect data extraction [ACS13]. We mitigated potential threats to conclusion validity by applying various best practices and guidelines on systematic mapping studies and structured literature reviews (cf. [KC07, PFMM08, VBSN+09, PVK15]). The construct validity of the survey in P3 deals with the ability to conclude from survey data which may be threatened by drawing incorrect conclusions. To mitigate this type of threat, multiple researchers coded the data independently and subsequently compared the codes and drew conclusions together to avoid misinterpretation and misunderstanding of the data. Similar to the survey, the conclusion validity of the case studies conducted in P4, P8, P9, and P10 may be threatened by drawing incorrect conclusions. Hence, to mitigate risks related to the conclusion validity, we created a case study database comprising case study documents, e.g., audio recordings, interview protocols, and slide decks. The conclusion validity of P6 and P7 may be threatened by designing inconsistent solution artifacts. Multiple researchers were involved in their conceptualization by applying the recommendations by Buckl et al. [BMSS13a] to improve the conclusion validity of the proposed pattern language, patterns, and concepts.

CHAPTER 6

Conclusion and Future Research

This chapter concludes this dissertation and provides an outlook on future research. Specifically, Section 6.1 recapitulates the research questions raised in Section 1.2, and Section 6.2 outlines potential future research paths that are enabled by the dissertation's findings.

6.1. Conclusion

Driven by the distress of traditional software methods being unable to respond in a timely and flexible manner to changes that software projects are constantly facing [WC03, Ket07], the agile movement emerged in the 1990s, leading to the creation of various agile methods and the Agile Manifesto in 2001 [Ket07]. Given the successful application of agile methods in small projects, companies are encouraged to scale them to larger endeavors [DFI14, AR16]. As is often the case with emerging phenomena in software engineering, the practice has led to research in the field of large-scale agile development, primarily due to the efforts of practitioners and consultants [Paa17, EWC21]. While this practice-driven phenomenon certainly has its merits [EWC21], the nascent state of empirical studies [DPL16, DFP19] and the presence of several research gaps [UPP+22] do not provide the anticipated assistance that practitioners expect. Motivated by this backdrop, the purpose of this dissertation was to contribute to the existing body of knowledge by empirically analyzing the large-scale adoption of agile methods and addressing four research gaps: (i) missing overview of the state of research, (ii) lack of overview and analysis of scaling agile frameworks, (iii) missing identification and documentation of patterns, and (iv) lack of investigation related to the collaboration between architects and agile teams.

Conclusions related to RQ1. This dissertation provides an analysis of existing research on large-scale agile development, covering 136 publications from 2007 to 2019, and sheds light on the meaning of "*large-scale agile development*" by investigating the development efforts of the

reported companies. This dissertation provides a better understanding of the nature of the research field by examining research trends and systematically classifying existing studies. It also reveals the intellectual structure of the research area by exposing its seminal works. This dissertation also assesses the state of the art and maturity level of the research field by identifying central research themes and outlining a research agenda for future research efforts. The obtained results allow researchers to get a sound introduction to the present research and serve as a starting point for their new research efforts. The gathered findings enable practitioners to develop a common terminology, understanding, and perspective on large-scale agile development and identify research they can use to solve their specific real-world problems.

Conclusions related to RQ2. This dissertation compiles a list of 22 scaling frameworks and assesses their maturity using common comparison criteria. This dissertation improves the overall understanding of these frameworks by exploring the main reasons for their emergence and identifying their adoption's main benefits and challenges from the perspective of their inventors. This dissertation also provides empirical evidence of the adoption of scaling frameworks in practice by describing the application of the mature scaling framework LeSS in four different products of a German automotive manufacturer. In addition to empirically studying the most popular frameworks, SAFe, LeSS, and DAD, researchers can use our results to explore the real-world application of the remaining frameworks that have not been sufficiently studied. Researchers can build on our results to quantitatively assess and validate the benefits and challenges of adopting scaling frameworks and use our results to develop and strengthen theories about applying these frameworks in the industry. As practitioners find choosing a particular framework problematic due to the lack of evaluation models for comparing these frameworks, they can use our results to make informed decisions when selecting scaling frameworks. Our results offer practitioners several lessons learned to consider when adopting these frameworks.

Conclusions related to RQ3. This dissertation points to a wide range of challenges that companies have to reckon with when scaling agile methods by providing a list of 79 challenges. In this light, this dissertation proposes a set of patterns, particularly for agile coaches and scrum masters, to address their typical challenges. Researchers can use our findings to determine which identified difficulties are encountered in practice and accordingly suggest new best practices for tackling them. Researchers can also utilize our results to assist companies in selecting and adopting our proposed patterns to solve their concerns. Practitioners can use the identified problems as a single resource for guidance and employ the suggested patterns as a reference for addressing their challenges in adopting agile methods at scale.

Conclusions related to RQ4. This dissertation sheds light on the tension between architecture and agility by providing insights related to the collaboration between architects and agile teams in large-scale agile development efforts. Researchers can use our results to develop new theories and explain observed phenomena related to the collaboration between architects and agile teams. Practitioners can use our findings to compare their current setups with the cases presented and explore how to achieve a fruitful collaboration between architects and agile teams. Practitioners can learn from our results and become aware of the responsibilities of enterprise architects and what agile teams expect from enterprise architects so that both can be more tolerant of each other and collaborate better together.

6.2. Outlook

In the following, we outline several avenues for future research for each of the research questions scientists can use as a starting point for new research efforts (see Table 6.1).

Table 6.1. Avenues for future research building on the embedded publications

RQ	No.	Potential research avenues
RQ1	P1	• Creation of rigorously developed frameworks, methods, and tools • Development of conceptual models and theories • Performing research on new emerging research themes • Using the research agenda to address open research gaps
RQ2	P2	• Usage of quantitative tools to assess strengths and weaknesses of scaling frameworks • Performing cross-case analyses to compare the adoption scaling frameworks • Conducting explanatory studies on how contextual factors affect the selection of a scaling framework
	P3	
	P4	
RQ3	P5	• Publication of a pattern catalog comprising stakeholder-related concerns and patterns • Implementation of the proposed patterns in the industry
	P6	
	P7	
RQ4	P8	• Performing explanatory studies on the relations between a company's agile maturity and the collaboration between architects and agile teams • Conducting longitudinal studies on how agile teams' expectations for enterprise architects change over time
	P9	
	P10	

Future research avenues related to RQ1. The current state of research in large-scale agile development shows that most studies tend to be observational, describing how companies apply agile practices at scale. None of the existing studies have contributed to developing new or improved tools [UPP+22]. Based on this observation, we encourage researchers to create rigorously developed frameworks, methods, and tools to meet practitioners' needs. There is also an apparent lack of the meaning of the term "*large-scale agile development*" [UPP+22], which hinders effective collaboration and progress in the research discipline. We recommend that scientists should provide more conceptual clarity on this term, which plays a crucial role in advancing the research field [DFI14]. While most case studies on large-scale agile development tend to be exploratory and less theoretical, future studies should establish a solid theoretical scaffold for the observed phenomena, similar to those on agile software development (cf. [DNBM12]). Early research started with contributions to global and distributed software engineering and scaling agile frameworks and continued with investigations related to the communication and coordination of agile teams [UPP+22]. Recently, themes such as team autonomy and large-scale agile transformations have gained scholarly interest, as they are still highly neglected but of great importance. We recommend that future studies should use our research agenda (cf. P1) to address research gaps in these two emerging research topics. We have compiled a list of 81 research questions (cf. P1) that researchers can use to address the outstanding research gaps.

Future research avenues related to RQ2. As most studies related to scaling frameworks are primarily qualitative and descriptive (cf. P2, P3, P4), and there is a lack of quantitative

and explanatory studies [UPP+22], we made the first attempt with two additional publications. These quantitatively assess the reasons, expected benefits, and satisfaction levels of practitioners of adopting scaling frameworks (cf. [PUH+21]) and measure the benefits and challenges of adopting SAFe (cf. [PUPH21]). We encourage researchers to build on these results and conduct further studies to evaluate the strengths and weaknesses of scaling frameworks quantitatively. Adopting a scaling framework may require some time and effort [EWC21] and high initial investment costs before it pays off [CC19b]. Given the absence of comparison models to guide practitioners in adopting a particular framework [CC19b], we encourage researchers to examine how contextual factors affect framework selection and conduct cross-case analyses to compare their adoption across companies using common comparative criteria. The insights gained can lead to a comparison model that practitioners can use to select a framework.

Future research avenues related to RQ3. Due to the limited scope of scientific papers, two embedded publications presented only four patterns for demonstrating the proposed pattern language and another five patterns for agile coaches and scrum masters in their entirety (cf. P6, P7). One additional publication outside this dissertation's scope presented five other patterns for enterprise and solution architects (cf. [UM20b]). Since the given patterns represent only a fraction of the entire collection, the dissertation can be further extended by publishing a pattern catalog comprising 60 patterns based on the current knowledge base. While the proposed patterns currently reflect a conceptualization of the observations made by the researcher, future research endeavors should select and configure them for their application in companies, as well as observe their actual instantiations and identify possible deviations from their initial adoption. In this way, future research efforts can close the research activity cycle of the pattern-based design research method [BMSS13a] and enrich the current collection with new or revised patterns.

Future research avenues related to RQ4. Since a company's agile maturity has a decisive impact on its stakeholders' way of working [vMvV14, Laa17, SvECV21], researchers should conduct explanatory studies to show potential interrelations between a company's agile maturity and the collaboration between agile teams and architects. These studies would be a meaningful extension of our research as the companies that participated in our case studies tended to be more traditional and therefore sought to acquire an agile mindset as part of their agile transformations. Future studies should contrast both research results and compare how architects and agile teams work together in both more agile-mature companies and traditional companies. As large-scale agile transformations represent extensive episodic change processes [FH18], agile teams' expectations of enterprise architects may fluctuate over time. Although we followed and observed the participating companies in our case studies over a period, future studies should conduct longitudinal studies and observe how agile teams' expectations of enterprise architects change over time. For example, researchers should describe which stages of the transformations agile teams typically have which expectations to enterprise architects, e.g., at the beginning of these transformations, enterprise architects might be expected to outline the overall direction of the ongoing efforts and then be expected to support the endeavors in making technology- and architecture-related decisions during the ongoing change process.

Bibliography

[AAH11] Alireza Abbasi, Jörn Altmann, and Liaquat Hossain. Identifying the effects of co-authorship networks on the performance of scholars: A correlation and regression analysis of performance measures and social network analysis measures. *Journal of Informetrics (JOI)*, 5(4):594–607, 2011.

[AASW17] Faiza Anwer, Shabib Aftab, Muhammad Shah, and Usman Waheed. Comparative analysis of two popular agile process models: Extreme programming and scrum. *International Journal of Computer Science and Telecommunications (IJCST)*, 8(2):1–7, 2017.

[ABB98] Alison Anderson, Ralph Beattie, and Kent Beck. Chrysler goes to "extremes". *Distributed Computing*, 1(10):24–28, 1998.

[ABK10] Pekka Abrahamsson, Muhammad Ali Babar, and Philippe Kruchten. Agility and architecture: Can they coexist? *IEEE Software*, 27(2):16–22, 2010.

[ACS13] Apostolos Ampatzoglou, Sofia Charalampidou, and Ioannis Stamelos. Research state of the art on gof design patterns: Mapping study. *Journal of Systems and Software (JSS)*, 86(7):1945–1964, 2013.

[ACW09] Pekka Abrahamsson, Kieran Conboy, and Xiaofeng Wang. 'lots done, more to do': The current state of agile systems development research. *European Journal of Information Systems (EJIS)*, 18(4):281–284, 2009.

[ÅF06] Pär Ågerfalk and Brian Fitzgerald. Old petunias in new bowls? *Communications of the ACM*, 49(10):27–34, 2006.

[AIS77] Christopher Alexander, Sara Ishikawa, and Murray Silverstein. *A Pattern Language: Towns, Buildings, Construction.* Oxford University Press, Oxford, England, UK, 1977.

[AL12] Scott Ambler and Mark Lines. *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise.* Pearson Education, Boston, MA, USA, 2012.

[AL18] Scott Ambler and Mark Lines. *Introduction to Disciplined Agile Delivery: A Small Agile Team's Journey from Scrum to DevOps.* CreateSpace, Scotts Valley, CA, USA, 2 edition, 2018.

[AL20] Scott Ambler and Mark Lines. *Choose your WoW: a Disciplined Agile Delivery Handbook for Optimizing your Way of Working.* Project Management Institute, Newtown Square, PA, USA, 2020.

[Ale64] Christopher Alexander. *Notes on the Synthesis of Form.* Harvard University Press, Cambridge, MA, USA, 1964.

[Amb07] Scott Ambler. Agile software development at scale. In *Proceedings of the 2nd Central and East European Conference on Software Engineering Techniques (CEE-SET)*, pages 1–12, Berlin, Heidelberg, Germany, 2007. Springer Berlin Heidelberg.

[AMG16] Samuil Angelov, Marcel Meesters, and Matthias Galster. Architects in scrum: What challenges do they face? In *Proceedings of the 10th European Conference on Software Architecture (ECSA)*, pages 229–237, Cham, Switzerland, 2016. Springer International.

[AMO13] Muhammad Ovais Ahmad, Jouni Markkula, and Markku Oivo. Kanban in software development: A systematic literature review. In *Proceedings of the 39th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 9–16, New York, NY, USA, 2013. IEEE.

[And10] David Anderson. *Kanban: Successful Evolutionary Change for your Technology Business.* Blue Hole Press, Sequim, WA, USA, 2010.

[And17] Stephen Andriole. The death of big software. *Communications of the ACM*, 60(12):29–32, 2017.

[AR16] Mashal Alqudah and Rozilawati Razali. A review of scaling agile methods in large software development. *International Journal on Advanced Science, Engineering and Information Technology (IJASEIT)*, 6(6):828–837, 2016.

[ASRW17] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, and Juhani Warsta. Agile software development methods: Review and analysis. *arXiv Preprint arXiv:1709.08439*, 2017.

[AST06] Ashish Agarwal, Ravi Shankar, and Manoy Kumar Tiwari. Modeling the metrics of lean, agile and leagile supply chain: An anp-based approach. *European Journal of Operational Research (EJOR)*, 173(1):211–225, 2006.

[AWSR03] Pekka Abrahamsson, Juhani Warsta, Mikko Siponen, and Jussi Ronkainen. New directions on agile methods: A comparative analysis. In *Proceedings of the 25th International Conference on Software Engineering (ICSE)*, pages 244–254, New York, NY, USA, 2003. IEEE.

[Bab09] Muhammad Ali Babar. An exploratory study of architectural practices and challenges in using agile software development approaches. In *Proceedings of the Joint 8th Working Conference on Software Architecture & 3rd European Conference on Software Architecture (WICSA/ECSA)*, pages 81–90, New York, NY, USA, 2009. IEEE.

[Bak00] Michael Baker. Writing a literature review. *The Marketing Review (TMR)*, 1(2):219–247, 2000.

[Bat20] Dinesh Batra. Research challenges and opportunities in conducting quantitative studies on large-scale agile methodology. *Journal of Database Management (JDM)*, 31(2):64–73, 2020.

[BBD09] Hilary Berger and Paul Beynon-Davies. The utility of rapid application development in large-scale, complex projects. *Information Systems Journal (ISJ)*, 19(6):549–570, 2009.

[BBM13] Muhammad Ali Babar, Alan Brown, and Ivan Mistrík. *Agile Software Architecture: Aligning Agile Processes and Software Architectures*. Elsevier, Amsterdam, Netherlands, 2013.

[BBS10] Jan Bosch and Petra Bosch-Sijtsema. Coordination between global agile teams: From process to architecture. In *Agility Across Time and Space: Implementing Agile Methods in Global Software Projects*, pages 217–233. Springer Berlin Heidelberg, Berlin, Heidelberg, Germany, 2010.

[BBvB+01] 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. `https://agilemanifesto.org/`, 2001. [Online; accessed 03-04-2022].

[BCS+10] Mike Beedle, James Coplien, Jeff Sutherland, Jens Østergaard, Ademar Aguiar, and Ken Schwaber. Essential scrum patterns. In *Proceedings of the 14th European Conference on Pattern Languages of Programs (EuroPLoP)*, pages 1–17, Munich, Germany, 2010. The Hillside Group.

[BDS+99] Mike Beedle, Martine Devos, Yonat Sharon, Ken Schwaber, and Jeff Sutherland. Scrum: An extension pattern language for hyperproductive software development. In *Pattern Languages of Program Design*, volume 4, pages 637–651. Addison-Wesley, Boston, MA, USA, 1999.

[Bec00] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Boston, MA, USA, 2000.

[Ben83] Herbert Benington. Production of large computer programs. *Annals of the History of Computing (AHC)*, 5(4):350–361, 1983.

[BF12] Marie-Joëlle Browaeys and Sandra Fisser. Lean and agile: an epistemological reflection. *The Learning Organization*, 19(3):207–218, 2012.

[BGM87] Izak Benbasat, David Goldstein, and Melissa Mead. The case research strategy in studies of information systems. *MIS Quarterly (MISQ)*, pages 369–386, 1987.

[Bha12] Anol Bhattacherjee. *Social Science Research: Principles, Methods, and Practices.* CreateSpace, Scotts Valley, CA, USA, 2012.

[BHD07] Frank Buschmann, Kevlin Henney, and Schmidt Douglas. *Pattern Oriented Software Architecture: On Patterns and Pattern Languages*, volume 5. John Wiley & Sons, Hoboken, NJ, USA, 2007.

[BKB+07] Pearl Brereton, Barbara Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software (JSS)*, 80(4):571–583, 2007.

[BKNÖ14] Stephany Bellomo, Philippe Kruchten, Robert Nord, and Ipek Özkaya. How to agilely architect an agile architecture. *Cutter IT Journal*, 27(2):12–17, 2014.

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture: A System of Patterns*, volume 1. John Wiley & Sons, Hoboken, NJ, USA, 1996.

[BMSS13a] Sabine Buckl, Florian Matthes, Alexander Schneider, and Christian Schweda. Pattern-based design research – an iterative research method balancing rigor and relevance. In *Proceedings of the 8th International Conference on Design Science Research in Information Systems and Technology (DESRIST)*, pages 73–87, Berlin, Heidelberg, Germany, 2013. Springer Berlin Heidelberg.

[BMSS13b] Sabine Buckl, Florian Matthes, Alexander Schneider, and Christian Schweda. Pattern-based design research in enterprise architecture management. In *Proceedings of the 25th International Conference on Advanced Information Systems Engineering Workshops (CAiSEW)*, pages 30–42, Berlin, Heidelberg, Germany, 2013. Springer Berlin Heidelberg.

[BNO12] Felix Bachmann, Robert Nord, and Ipek Ozakaya. Architectural tactics to support rapid and agile stability. *CrossTalk*, 8:20–25, 2012.

[Boe02] Barry Boehm. Get ready for agile methods, with care. *Computer*, 35(1):64–69, 2002.

[BT03] Barry Boehm and Richard Turner. Using risk to balance agile and plan-driven methods. *Computer*, 36(6):57–66, 2003.

[BT05] Barry Boehm and Richard Turner. Management challenges to implementing agile processes in traditional development organizations. *IEEE Software*, 22(5):30–39, 2005.

[Buc11] Sabine Buckl. *Developing Organization-Specific Enterprise Architecture Management Functions Using a Method Base.* Dissertation, Technical University of Munich, Munich, Germany, 2011.

[BW96] Izak Benbasat and Ron Weber. Research commentary: Rethinking "diversity" in information systems research. *Information Systems Research (ISR)*, 7(4):389–399, 1996.

[BWR11] Elizabeth Bjarnason, Krzysztof Wnuk, and Björn Regnell. A case study on benefits and side-effects of agile practices in large-scale requirements engineering. In *Proceedings of the 1st Workshop on Agile Requirements Engineering (AREW)*, pages 1–5, New York, NY, USA, 2011. ACM.

[CC18] John Creswell and David Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE, Thousand Oaks, CA, USA, 2018.

[CC19a] Noel Carroll and Kieran Conboy. Applying normalization process theory to explain large-scale agile transformations. In *Proceedings of the 14th International Research Workshop on IT Project Management (IRWITPM)*, pages 1–11, Atlanta, GA, USA, 2019. AIS.

[CC19b] Kieran Conboy and Noel Carroll. Implementing large-scale agile frameworks: Challenges and recommendations. *IEEE Software*, 36(2):44–50, 2019.

[CCJ+18] Mert Canat, Núria Pol Català, Alexander Jourkovski, Svetlomir Petrov, Martin Wellme, and Robert Lagerström. Enterprise architecture and agile development: Friends or foes? In *Proceedings of the 22nd International Enterprise Distributed Object Computing Workshop (EDOCW)*, pages 176–183, New York, NY, USA, 2018. IEEE.

[CH04] James Coplien and Neil Harrison. *Organizational Patterns of Agile Software Development*. Addison-Wesley, Boston, MA, USA, 2004.

[Cho08] Juyun Cho. Issues and challenges of agile software development with scrum. *Issues in Information Systems*, 9(2):188–195, 2008.

[CL13] Bas Vodde Craig Larman. Scaling agile development. *CrossTalk*, 9:8–12, 2013.

[CLC04] David Cohen, Mikael Lindvall, and Patrícia Costa. An introduction to agile methods. In *Advances in Computers*, volume 62, pages 1–66. Elsevier, Amsterdam, Netherlands, 2004.

[CLVB03] Marcus Ciolkowski, Oliver Laitenberger, Sira Vegas, and Stefan Biffl. Practical experiences in the design and conduct of surveys in empirical software engineering. In *Empirical Methods and Studies in Software Engineering*, pages 104–128. Springer Berlin Heidelberg, Berlin, Heidelberg, Germany, 2003.

[Con09] Kieran Conboy. Agility from first principles: Reconstructing the concept of agility in information systems development. *Information Systems Research (ISR)*, 20(3):329–354, 2009.

[Coo88] Harris Cooper. Organizing knowledge syntheses: A taxonomy of literature reviews. *Knowledge in Society*, 1(1):104–126, 1988.

[Coo16] Robert Cooper. Agile-stage-gate hybrids: The next stage for product development blending agile and stage-gate methods can provide flexibility, speed, and improved communication in new-product development. *Research-Technology Management (RTM)*, 59(1):21–29, 2016.

[Cop96] James Coplien. *Software Patterns: Management Briefs*. Cambridge University Press, Cambridge, England, UK, 1996.

[CS16] Robert Cooper and Anita Sommer. The agile-stage-gate hybrid model: a promising new approach and a new research opportunity. *Journal of Product Innovation Management (JPIM)*, 33(5):513–526, 2016.

[CWR13] Oisín Cawley, Xiaofeng Wang, and Ita Richardson. Lean software development – what exactly are we talking about? In *Proceedings of the 4th International Conference on Lean Enterprise Software and Systems (LESS)*, pages 16–31, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[DD08] Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology (IST)*, 50(9):833–859, 2008.

[DD09] Tore Dybå and Torgeir Dingsøyr. What do we know about agile software development? *IEEE Software*, 26(5):6–9, 2009.

[DFI14] Torgeir Dingsøyr, Tor Erlend Fægri, and Juha Itkonen. What is large in large-scale? a taxonomy of scale for agile software development. In *Proceedings of the 15th International Conference on Product-Focused Software Process Improvement (PROFES)*, pages 273–276, Cham, Switzerland, 2014. Springer International.

[DFP19] Torgeir Dingsøyr, Davide Falessi, and Ken Power. Agile development at scale: the next frontier. *IEEE Software*, 36(2):30–38, 2019.

[DHL+08] Tom DeMarco, Peter Hruschka, Tim Lister, Suzanne Robertson, James Robertson, and Steve McMenamin. *Adrenaline Junkies and Template Zombies: Understanding Patterns of Project Behavior*. Dorset House, New York, NY, USA, 2008.

[Dig20] Digital.ai. 14th State of Agile Survey. `https://www.qagile.pl/wp-content/uploads/2020/06/14th-annual-state-of-agile-report.pdf`, 2020. [Online; accessed 03-04-2022].

[Dig21] Digital.ai. 15th State of Agile Survey. `https://digital.ai/resource-center/analyst-reports/state-of-agile-report`, 2021. [Online; accessed 03-04-2022].

[DM13] Torgeir Dingsøyr and Nils Brede Moe. Research challenges in large-scale agile software development. *ACM SIGSOFT Software Engineering Notes*, 38(5):38–39, 2013.

[DM14] Torgeir Dingsøyr and Nils Brede Moe. Towards principles of large-scale agile development. In *Proceedings of the 15th International Conference on Agile Software Development Scientific Workshops (AGILESW)*, pages 1–8, Cham, Switzerland, 2014. Springer International.

[DMFS18] Torgeir Dingsøyr, Nils Brede Moe, Tor Erlend Fægri, and Eva Amdahl Seim. Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation. *Empirical Software Engineering (EMSE)*, 23(1):490–520, 2018.

[DMO18] Torgeir Dingsøyr, Nils Brede Moe, and Helena Holmström Ohlsson. Towards an understanding of scaling frameworks and business agility: A summary of the 6th international workshop at xp2018. In *Proceedings of the 19th International Conference on Agile Software Development Scientific Workshops (AGILESW)*, pages 1–4, New York, NY, USA, 2018. ACM.

[DNBM12] Torgeir Dingsøyr, Sridhar Nerur, Venugopal Balijepally, and Nils Brede Moe. A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software (JSS)*, 85(6):1213–1221, 2012.

[DPL16] 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 (JSS)*, 119:87–108, 2016.

[DSHT17] Paul Drews, Ingrid Schirmer, Bettina Horlach, and Carsten Tekaat. Bimodal enterprise architecture management: The emergence of a new eam function for a bizdevops-based fast it. In *Proceedings of the 21st International Enterprise Distributed Object Computing Workshop (EDOCW)*, pages 57–64, New York, NY, USA, 2017. IEEE.

[DSRB00] Schmidt Douglas, Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*, volume 2. John Wiley & Sons, Hoboken, NJ, USA, 2000.

[DST18] Philipp Diebold, Anna Schmitt, and Sven Theobald. Scaling agile: How to select the most appropriate framework. In *Proceedings of the 19th International Conference on Agile Software Development Scientific Workshops (AGILESW)*, pages 1–4, New York, NY, USA, 2018. ACM.

[EAO12] Christof Ebert, Pekka Abrahamsson, and Nilay Oza. Lean software development. *IEEE Software*, 29(05):22–25, 2012.

[EP17] Christof Ebert and Maria Paasivaara. Scaling agile. *IEEE Software*, 34(6):98–103, 2017.

[Ern10] Alexander Ernst. *A Pattern-based Approach to Enterprise Architecture Management.* Dissertation, Technical University of Munich, Munich, Germany, 2010.

[ESSD08] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer London, London, England, UK, 2008.

[EWC21] Henry Edison, Xiaofeng Wang, and Kieran Conboy. Comparing methods for large-scale agile software development: A systematic literature review. *IEEE Transactions on Software Engineering (TSE)*, pages 1–23, 2021. in press.

[FH18] Christoph Fuchs and Thomas Hess. Becoming agile in the digital transformation: The process of a large-scale agile transformation. In *Proceedings of the 39th International Conference on Information Systems (ICIS)*, pages 1–17, Atlanta, GA, USA, 2018. AIS.

[FJ13] Floyd Fowler Jr. *Survey Research Methods*. SAGE, Thousand Oaks, CA, USA, 2013.

[Fli92] Uwe Flick. Triangulation revisited: Strategy of validation or alternative? *Journal for the Theory of Social Behaviour (JTSB)*, 22(2):175–197, 1992.

[FMS14] Brian Fitzgerald, Mariusz Musiał, and Klaas-Jan Stol. Evidence-based decision making in lean software project management. In *Proceedings of the 36th International Conference on Software Engineering Workshops (ICSEW)*, pages 93–102, New York, NY, USA, 2014. ACM.

[Fow06] Martin Fowler. Writing software patterns. `https://www.martinfowler.com/articles/writingPatterns.html`, 2006. [Online; accessed 03-04-2022].

[Fow13] Martin Fowler. Extreme programming. `https://martinfowler.com/bliki/ExtremeProgramming.html`, 2013. [Online; accessed 03-04-2022].

[Fre78] Linton Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1(3):215–239, 1978.

[FS10] Sallyann Freudenberg and Helen Sharp. The top 10 burning research questions from practitioners. *IEEE Software*, 27(5):8–9, 2010.

[FSOO13] Brian Fitzgerald, Klaas-Jan Stol, Ryan O'Sullivan, and Donal O'Brien. Scaling agile methods to regulated environments: An industry case study. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, pages 863–872, New York, NY, USA, 2013. IEEE.

[FSS02] Robert Futrell, Donald Shafer, and Linda Shafer. *Quality Software Project Management*. Prentice-Hall, Hoboken, NJ, USA, 2002.

[FV06] Ann Fruhling and Gert-Jan De Vreede. Field experiences with extreme programming: Developing an emergency response system. *Journal of Management Information Systems (JMIS)*, 22(4):39–68, 2006.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston, MA, USA, 1994.

[GJJG11] Cristina Venera Geambaşu, Iulia Jianu, Ionel Jianu, and Alexandru Gavrilă. Influence factors for the choice of a software development methodology. *Journal of Accounting and Management Information Systems (JAMIS)*, 10(4):479–494, 2011.

[GPRN18] Ahmad Nauman Ghazi, Kai Petersen, Sri Sai Vijay Raj Reddy, and Harini Nekkanti. Survey research in software engineering: Problems and mitigation strategies. *IEEE Access*, 7:24703–24718, 2018.

[GS05] Christina Guo and Yesenia Sanchez. Workplace communication. In *Organizational Behavior in Health Care*, volume 2, pages 77–110. Jones and Bartlett Publishers, Sudbury, MA, USA, 2005.

[Gus18] Tomas Gustavsson. Practices for vertical and horizontal coordination in the scaled agile framework. In *Proceedings of the 27th International Conference on Information Systems Development (ISD)*, pages 1–12, Atlanta, GA, USA, 2018. AIS.

[GvLG21] Hong Guo, Darja Šmite, Jingyue Li, and Shang Gao. Enterprise architecture and agility: A systematic mapping study. In *Proceedings of the 11th International Symposium on Business Modeling and Software Design (BMSD)*, pages 296–305, Cham, Switzerland, 2021. Springer International.

[HA13] Amani Mahdi Mohammed Hamed and Hisham Abushama. Popular agile approaches in software development: Review and analysis. In *Proceedings of the 2013 International Conference on Computing, Electrical and Electronic Engineering (ICCEEE)*, pages 160–166, New York, NY, USA, 2013. IEEE.

[HATB95] Joseph Hair, Rolph Anderson, Ronald Tatham, and William Black. *Multivariate Data Analysis*. Prentice-Hall, Hoboken, NJ, USA, 4 edition, 1995.

[HATG21] Steffen Herbold, Aynur Amirfallah, Fabian Trautsch, and Jens Grabowski. A systematic mapping study of developer social network research. *Journal of Systems and Software (JSS)*, 171:1–20, 2021.

[Hau01] Jim Haungs. Pair programming on the c3 project. *Computer*, 34(2):118–119, 2001.

[HBP09] Emam Hossain, Muhammad Ali Babar, and Hye-Young Paik. Using scrum in global software development: A systematic literature review. In *Proceedings of the 4th International Conference on Global Software Engineering (ICGSE)*, pages 175–184, New York, NY, USA, 2009. IEEE.

[HC01] Jim Highsmith and Alistair Cockburn. Agile software development: the business of innovation. *Computer*, 34(9):120–127, 2001.

[HDSD20] Bettina Horlach, Andreas Drechsler, Ingrid Schirmer, and Paul Drews. Everyone's going to be an architect: Design principles for architectural thinking in agile organizations. In *Proceedings of the 53rd Hawaii International Conference on System Sciences (HICSS)*, pages 1–10, Honolulu, HI, USA, 2020. ScholarSpace.

[HEK15] Sebastian Hanschke, Jan Ernsting, and Herbert Kuchen. Integrating agile software development and enterprise architecture management. In *Proceedings of the 48th Hawaii International Conference on System Sciences (HICSS)*, pages 4099–4108, New York, NY, USA, 2015. IEEE.

[HH02] James Highsmith and Jim Highsmith. *Agile Software Development Ecosystems.* Addison-Wesley, Boston, MA, USA, 2002.

[HMPR04] Alan Hevner, Salvatore March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly (MISQ)*, 28(1):75–105, 2004.

[HOV05] Richard Hightower, Warner Onstine, and Paul Visan. *Professional Java Tools for Extreme Programming.* John Wiley & Sons, Hoboken, NJ, USA, 2005.

[HPLE13] Ville Heikkilä, Maria Paasivaara, Casper Lassenius, and Christian Engblom. Continuous release planning in a large-scale scrum development organization at ericsson. In *Proceedings of the 14th International Conference on Agile Software Development (AGILE)*, pages 195–209, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[HRSM14] Matheus Hauder, Sascha Roth, Christopher Schulz, and Florian Matthes. Agile enterprise architecture management: An analysis on the application of agile principles. In *Proceedings of the 4th International Symposium on Business Modeling and Software Design (BMSD)*, pages 38–46, Setúbal, Portugal, 2014. SciTePress.

[HSG18] Rashina Hoda, Norsaremah Salleh, and John Grundy. The rise and evolution of agile software development. *IEEE Software*, 35(5):58–63, 2018.

[HvM11] Geir Hanssen, Darja Šmite, and Nils Brede Moe. Signs of agile trends in global software engineering research: A tertiary study. In *Proceedings of the 6th International Conference on Global Software Engineering Workshop (GSEW)*, pages 17–23, New York, NY, USA, 2011. IEEE.

[IM95] Stephen Isaac and William Michael. *Handbook in Research and Evaluation: A Collection of Principles, Methods, and Strategies useful in the Planning, Design, and Evaluation of Studies in Education and the Behavioral Sciences.* Edits Publishers, 3 edition, 1995.

[ISM+15] Irum Inayat, Siti Salim, Sabrina Marczak, Maya Daneva, and Shahaboddin Shamshirband. A systematic literature review on agile requirements engineering practices and challenges. *Computers in Human Behavior*, 51:915–929, 2015.

[JOT07] Burke Johnson, Anthony Onwuegbuzie, and Lisa Turner. Toward a definition of mixed methods research. *Journal of Mixed Methods Research (JMMR)*, 1(2):112–133, 2007.

[Kas05] Mark Kasunic. Designing an effective survey. Technical report, Carnegie-Mellon University, Pittsburgh, PA, USA, 2005.

[KB13] Barbara Kitchenham and Pearl Brereton. A systematic review of systematic review process research in software engineering. *Information and Software Technology (IST)*, 55(12):2049–2075, 2013.

[KBB10] Barbara Kitchenham, David Budgen, and Pearl Brereton. The value of mapping studies – a participant-observer case study. In *Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 1–9, Swindon, England, UK, 2010. BCS Learning & Development.

[KBB11] Barbara Kitchenham, David Budgen, and Pearl Brereton. Using mapping studies as the basis for further research – a participant-observer case study. *Information and Software Technology (IST)*, 53(6):638–651, 2011.

[KBB15] Barbara Ann Kitchenham, David Budgen, and Pearl Brereton. *Evidence-based Software Engineering and Systematic Reviews*, volume 4. CRC Press, Boca Raton, FL, USA, 2015.

[KC07] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. Technical report, Keele University, Keele, England, UK, 2007.

[KDJ04] Barbara Kitchenham, Tore Dybå, and Magne Jorgensen. Evidence-based software engineering. In *Proceedings of the 26th International Conference on Software Engineering (ICSE)*, pages 273–281, New York, NY, USA, 2004. IEEE.

[Kee02] Gerold Keefer. Extreme programming considered harmful for reliable software development. Technical report, AVOCA GmbH, Kapellen-Drusweiler, Germany, 2002.

[Kel12] Allan Kelly. *Business Patterns for Software Developers*. John Wiley & Sons, Hoboken, NJ, USA, 2012.

[Ket07] Petri Kettunen. Extending software project agility with new product development enterprise agility. *Software Process: Improvement and Practice (SPIP)*, 12(6):541–548, 2007.

[KHR18] Martin Kalenda, Petr Hyna, and Bruno Rossi. Scaling agile in large organizations: Practices, challenges, and success factors. *Journal of Software: Evolution and Process (JSEP)*, 30(10):1–25, 2018.

[Kit04] Barbara Kitchenham. Procedures for performing systematic reviews. Technical report, Keele University, Keele, England, UK, 2004.

[KL17] Daryl Kulak and Hong Li. *The Journey to Enterprise Agility: Systems Thinking and Organizational Legacy*. Springer Nature, Cham, Switzerland, 2017.

[KL20] Richard Knaster and Dean Leffingwell. *SAFe 5.0 Distilled: Achieving Business Agility with the Scaled Agile Framework*. Addison-Wesley, Boston, MA, USA, 2020.

[KMI15] Eetu Kupiainen, Mika Mäntylä, and Juha Itkonen. Using metrics in agile and lean software development–a systematic literature review of industrial studies. *Information and Software Technology (IST)*, 62:143–163, 2015.

[KML20] Dina Koutsikouri, Sabine Madsen, and Nataliya Berbyuk Lindström. Agile transformation: How employees experience and cope with transformative change. In *Proceedings of the 21st International Conference on Agile Software Development Scientific Workshops (AGILESW)*, pages 155–163, Cham, Switzerland, 2020. Springer International.

[KP08] Barbara Kitchenham and Shari Pfleeger. Personal opinion surveys. In *Guide to Advanced Empirical Software Engineering*, pages 63–92. Springer London, London, England, UK, 2008.

[KPP+02] Barbara Kitchenham, Shari Lawrence Pfleeger, Lesley Pickard, Peter Jones, David Hoaglin, Khaled El Emam, and Jarrett Rosenberg. Guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering (TSE)*, 28(8):721–734, 2002.

[Laa17] Maarit Laanti. Agile transformation model for large software development organizations. In *Proceedings of the 18th International Conference on Agile Software Development Scientific Workshops (AGILESW)*, pages 1–5, New York, NY, USA, 2017. ACM.

[Lad09] Corey Ladas. *Scrumban-Essays on Kanban Systems for Lean Software Development*. Modus Cooperandi Press, Seattle, WA, USA, 2009.

[LB03] Craig Larman and Victor Basili. Iterative and incremental developments: A brief history. *Computer*, 36(6):47–56, 2003.

[LE06] Yair Levy and Timothy Ellis. A systems approach to conduct an effective literature review in support of information systems research. *Informing Science Journal (InformingSciJ)*, 9:181–212, 2006.

[LeS22] LeSS Company. Less framework. `https://less.works/less/framework`, 2022. [Online; accessed 03-04-2022].

[LGJO17] Howard Lei, Farnaz Ganjeizadeh, Pradeep Kumar Jayachandran, and Pinar Ozcan. A statistical analysis of the effects of scrum and kanban on software development projects. *Robotics and Computer-Integrated Manufacturing (RCIM)*, 43:59–67, 2017.

[Lik03] Jeffrey Liker. *The Toyota Way*. McGraw-Hill Professional, New York, NY, USA, 2003.

[LMD+04] Mikael Lindvall, Dirk Muthig, Aldo Dagnino, Christina Wallin, Michael Stupperich, David Kiefer, John May, and Tuomo Kahkonen. Agile software development in large organizations. *Computer*, 37(12):26–34, 2004.

[LMZ08] Dean Leffingwell, Ryan Martens, and Mauricio Zamora. Principles of agile architecture. Technical report, Leffingwell and Rally Software Development, Boston, MA, USA, 2008.

[LSA11] Maarit Laanti, Outi Salo, and Pekka Abrahamsson. Agile methods rapidly replacing traditional methods at nokia: A survey of opinions on agile transformation. *Information and Software Technology (IST)*, 53(3):276–290, 2011.

[LSE+13] Lina Lagerberg, Tor Skude, Pär Emanuelsson, Kristian Sandahl, and Daniel Ståhl. The impact of agile principles and practices on large-scale software development projects: A multiple-case study of two projects at ericsson. In *Proceedings of the 7th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 348–356, New York, NY, USA, 2013. IEEE.

[LSMdMH15] Johan Linåker, Sardar Muhammad Sulaman, Rafael Maiani de Mello, and Martin Höst. Guidelines for conducting surveys in software engineering. Technical report, Lund University, Lund, Sweden, 2015.

[LSS13] Lawrence Locke, Waneen Wyrick Spirduso, and Stephen Silverman. *Proposals that work: A Guide for Planning Dissertations and Grant Proposals*. SAGE, Thousand Oaks, CA, USA, 2013.

[Lun11] Fred Lunenburg. Network patterns and analysis: Underused sources to improve communication effectiveness. *National Forum Of Educational Administration and Supervision Journal (NFEASJ)*, 28(4):1–27, 2011.

[LV16] Craig Larman and Bas Vodde. *Large-Scale Scrum: More with LeSS*. Addison-Wesley, Boston, MA, USA, 2016.

[Map09] Chuck Maples. Enterprise agile transformation: The two-year wall. In *Proceedings of the 2009 Agile Conference*, pages 90–95, New York, NY, USA, 2009. IEEE.

[McB03] Pete McBreen. *Questioning Extreme Programming*. Pearson Education, Boston, MA, USA, 2003.

[MD97] Gerard Meszaros and Jim Doble. A pattern language for pattern writing. In *Pattern Languages of Program Design*, volume 3, pages 529–574. Addison-Wesley, Boston, MA, USA, 1997.

[MD17] Nils Brede Moe and Torgeir Dingsøyr. Emerging research themes and updated research agenda for large-scale agile development: A summary of the 5th international workshop at xp2017. In *Proceedings of the 18th International Conference on Agile Software Development Scientific Workshops (AGILESW)*, pages 1–4, New York, NY, USA, 2017. ACM.

[MH94] Matthew Miles and Michael Huberman. *Qualitative Data Analysis: An Expanded Sourcebook*. SAGE, Thousand Oaks, CA, USA, 1994.

[Mil56] George Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63(2):81–97, 1956.

[MJ10] Neil Maiden and Sara Jones. Agile requirements can we have our cake and eat it too? *IEEE Software*, 27(3):87–88, 2010.

[MJ11] Peter Middleton and David Joyce. Lean software management: Bbc worldwide case study. *IEEE Transactions on Engineering Management (IEEE-TEM)*, 59(1):20–32, 2011.

[MKK10] Subhas Chandra Misra, Vinod Kumar, and Uma Kumar. Identifying some critical changes required in adopting agile practices in traditional software development projects. *International Journal of Quality & Reliability Management (IJQRM)*, 27(4):451–474, 2010.

[MNS12] Andrey Maglyas, Uolevi Nikula, and Kari Smolander. Lean solutions to software product management problems. *IEEE Software*, 29(5):40–46, 2012.

[Moc09] Martin Mocker. What is complex about 273 applications? untangling application architecture complexity in a case of european investment banking. In *Proceedings of the 42nd Hawaii International Conference on System Sciences*, pages 1–14, New York, NY, USA, 2009. IEEE.

[MRTR05] Hilkka Merisalo-Rantanen, Tuure Tuunanen, and Matti Rossi. Is extreme programming just old wine in new bottles: A comparison of two cases. *Journal of Database Management (JDM)*, 16(4):41–61, 2005.

[MS05] Peter Middleton and James Sutton. *Lean Software Strategies: Proven Techniques for Managers and Developers*. CRC Press, Boca Raton, FL, USA, 2005.

[Mye97] Michael Myers. Qualitative research in information systems. *MIS Quarterly (MISQ)*, 21(2):241–242, 06 1997.

[NMM05] Sridhar Nerur, RadhaKanta Mahapatra, and George Mangalaraj. Challenges of migrating to agile methodologies. *Communications of the ACM*, 48(5):72–78, 2005.

[Nob98] James Noble. Classifying relationships between object-oriented design patterns. In *Proceedings of the 11th Australian Software Engineering Conference (ASWEC)*, pages 98–107, New York, NY, USA, 1998. IEEE.

[NÖK14] Robert Nord, Ipek Özkaya, and Philippe Kruchten. Agile in distress: Architecture to the rescue. In *Proceedings of the 15th International Conference on Agile Software Development Scientific Workshops (AGILESW)*, pages 43–57, Cham, Switzerland, 2014. Springer International.

[NÖS12] Robert Nord, Ipek Özkaya, and Raghvinder Sangwan. Making architecture visible to improve flow management in lean software development. *IEEE Software*, 29(5):33–39, 2012.

[NRN08] Sridhar Nerur, Abdul Rasheed, and Vivek Natarajan. The intellectual structure of the strategic management field: An author co-citation analysis. *Strategic Management Journal (SMJ)*, 29(3):319–336, 2008.

[Oat11] Briony Oates. Evidence-based information systems: A decade later. In *Proceedings of the 19th European Conference on Information Systems (ECIS)*, pages 1–11, Atlanta, GA, USA, 2011. AIS.

[OBS06] Eric Overby, Anandhi Bharadwaj, and Vallabh Sambamurthy. Enterprise agility and the enabling role of information technology. *European Journal of Information Systems (EJIS)*, 15(2):120–131, 2006.

[Ohn88] Taiichi Ohno. *Toyota Production System: Beyond Large-scale Production*. CRC Press, Boca Raton, FL, USA, 1988.

[Paa17] Maria Paasivaara. Adopting safe to scale agile in a globally distributed organization. In *Proceedings of the 12th International Conference on Global Software Engineering (ICGSE)*, pages 36–40, New York, NY, USA, 2017. IEEE.

[PCL+04] Stanley Presser, Mick Couper, Judith Lessler, Elizabeth Martin, Jean Martin, Jennifer Rothgeb, and Eleanor Singer. Methods for testing and evaluating survey questions. *Public Opinion Quarterly (POQ)*, 68(1):109–130, 2004.

[PDL08] Maria Paasivaara, Sandra Durasiewicz, and Casper Lassenius. Using scrum in a globally distributed project: a case study. *Software Process: Improvement and Practice (SPIP)*, 13(6):527–544, 2008.

[Pet11] Kai Petersen. Is lean agile and agile lean? a comparison between two software development paradigms. In *Modern Software Engineering Concepts and Practices: Advanced Approaches*, pages 19–46. IGI Global, Hershey, PA, USA, 2011.

[PFMM08] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 68–77, Swindon, England, UK, 2008. BCS Learning & Development.

[PHK17] Jan Pries-Heje and Malene Krohn. The safe way to the agile organization. In *Proceedings of the 18th International Conference on Agile Software Development Scientific Workshops (AGILESW)*, pages 1–4, New York, NY, USA, 2017. ACM.

[PL16] Maria Paasivaara and Casper Lassenius. Challenges and success factors for large-scale agile transformations: A research proposal and a pilot study. In *Proceedings of the 17th International Conference on Agile Software Development Scientific Workshops (AGILESW)*, pages 1–5, New York, NY, USA, 2016. ACM.

[Pow16] Daryl Powell. Lean engineer-to-order manufacturing. In *The Routledge Companion to Lean Management*, pages 308–323. Routledge, Milton Park, Abingdon-on-Thames, Oxfordshire, England, UK, 2016.

[PP03] Mary Poppendieck and Tom Poppendieck. *Lean Software Development: An Agile Toolkit*. Addison-Wesley, Boston, MA, USA, 2003.

[PPL18] Abheeshta Putta, Maria Paasivaara, and Casper Lassenius. Benefits and challenges of adopting the scaled agile framework (safe): Preliminary results from a multivocal literature review. In *Proceedings of the 19th International Conference on Product-Focused Software Process Improvement (PROFES)*, pages 334–351, Cham, Switzerland, 2018. Springer International.

[PPL19] Abheeshta Putta, Maria Paasivaara, and Casper Lassenius. How are agile release trains formed in practice? a case study in a large financial corporation. In *Proceedings of the 20th International Conference on Agile Software Development (AGILE)*, pages 154–170, Cham, Switzerland, 2019. Springer International.

[Pro22] Project Management Institute. Full Delivery Life Cycles. `https://www.pmi.org/disciplined-agile/process/introduction-to-dad/full-delivery-lifecycles-introduction`, 2022. [Online; accessed 03-04-2022].

[PTRC07] Ken Peffers, Tuure Tuunanen, Marcus Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of Management Information Systems (JMIS)*, 24(3):45–77, 2007.

[PVK15] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology (IST)*, 64:1–18, 2015.

[PW10a] Kai Petersen and Claes Wohlin. The effect of moving from a plan-driven to an incremental software development approach with agile practices. *Empirical Software Engineering (EMSE)*, 15(6):654–693, 2010.

[PW10b] Kai Petersen and Claes Wohlin. Software process improvement through the lean measurement (spi-leam) method. *Journal of Systems and Software (JSS)*, 83(7):1275–1287, 2010.

[PW11] Kai Petersen and Claes Wohlin. Measuring the flow in lean software development. *Software: Practice and Experience (SPE)*, 41(9):975–996, 2011.

[Raj06] Vaclav Rajlich. Changing the paradigm of software engineering. *Communications of the ACM*, 49(8):67–70, 2006.

[Ram11] Pedro Pablo Ramos. *Network Models for Organizations: The Flexible Design of 21st Century Companies*. Palgrave Macmillan, New York, NY, USA, 2011.

[RB21] Daniel Remta and Alena Buchalcevova. Product owner's journey to safe – role changes in scaled agile framework. *Information*, 12(3):1–18, 2021.

[Rei09] Donald Reinertsen. *The Principles of Product Development Flow: Second Generation Lean Product Development*. Celeritas, Redondo Beach, CA, USA, 2009.

[RFDS16] Knut Rolland, Brian Fitzgerald, Torgeir Dingsøyr, and Klaas-Jan Stol. Problematizing agile in the large: Alternative assumptions for large-scale agile development. In *Proceedings of the 37th International Conference on Information Systems (ICIS)*, pages 1–21, Atlanta, GA, USA, 2016. AIS.

[RH09] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering (EMSE)*, 14(2):131–164, 2009.

[RHL+17] Pilar Rodríguez, Alireza Haghighatkhah, Lucy Ellen Lwakatare, Susanna Teppola, Tanja Suomalainen, Juho Eskeli, Teemu Karvonen, Pasi Kuvaja, June Verner, and Markku Oivo. Continuous deployment of software intensive products and services: A systematic mapping study. *Journal of Systems and Software (JSS)*, 123:263–291, 2017.

[RJ00] Linda Rising and Norman Janoff. The scrum software development process for small teams. *IEEE Software*, 17(4):26–32, 2000.

[RME03] Donald Reifer, Frank Maurer, and Hakan Erdoğmuş. Scaling agile methods. *IEEE Software*, 20(4):12–14, 2003.

[RMO+19] Pilar Rodríguez, Mika Mäntylä, Markku Oivo, Lucy Ellen Lwakatare, Pertti Seppänen, and Pasi Kuvaja. Advances in using agile and lean processes for software development. In *Advances in Computers*, volume 113, pages 135–224. Elsevier, Amsterdam, Netherlands, 2019.

[Rob02] Colin Robson. *Real World Research: A Resource for Social Scientists and Practitioner-Researchers*. Wiley-Blackwell, Hoboken, NJ, USA, 2002.

[RP14] Louis Rea and Richard Parker. *Designing and Conducting Survey Research: A Comprehensive Guide*. John Wiley & Sons, Hoboken, NJ, USA, 2014.

[RRN+18] Mohammad Abdur Razzak, Ita Richardson, John Noll, Clodagh Nic Canna, and Sarah Beecham. Scaling agile across the global organization: An early stage industrial safe self-assessment. In *Proceedings of the 13th International Conference on Global Software Engineering (ICGSE)*, pages 116–125, New York, NY, USA, 2018. IEEE.

[RS04] Jennifer Rowley and Frances Slack. Conducting a literature review. *Management Research News (MRN)*, 27(6):31–39, 2004.

[RS08] Don Rosenberg and Matt Stephens. *Extreme Programming Refactored: the Case Against XP*. Apress, New York, NY, USA, 2008.

[Rub12] Kenneth Rubin. *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Addison-Wesley, Boston, MA, USA, 2012.

[RV08] Michael Rosemann and Iris Vessey. Toward improving the relevance of information systems research to practice: the role of applicability checks. *MIS Quarterly (MISQ)*, 32(1):1–22, 2008.

[RWN+15] Dominik Rost, Balthasar Weitzel, Matthias Naab, Torsten Lenhart, and Hartmut Schmitt. Distilling best practices for agile development from architecture methodology. In *Proceedings of the 9th European Conference on Software Architecture (ECSA)*, pages 259–267, Cham, Switzerland, 2015. Springer International.

[SB19] Abdallah Salameh and Julian Bass. Spotify tailoring for promoting effectiveness in cross-functional autonomous squads. In *Proceedings of the 20th International Conference on Agile Software Development Scientific Workshops (AGILESW)*, pages 20–28, Cham, Switzerland, 2019. Springer International.

[SC90] Anselm Strauss and Juliet Corbin. *Basics of Qualitative Research*. SAGE, Thousand Oaks, CA, USA, 1990.

[Sca21] Scaled Agile. Reaching the Tipping Point. `https://www.scaledagileframework.com/reaching-the-tipping-point/`, 2021. [Online; accessed 03-04-2022].

[Sca22] Scaled Agile. SAFe 5 for Lean Enterprises. `https://www.scaledagileframework.com/safe-for-lean-enterprises/`, 2022. [Online; accessed 03-04-2022].

[Sch95] Ken Schwaber. Scrum development process. In *Proceedings of the 10th Conference on Objected-Oriented Programming, Systems, Languages, and Applications Workshops (OOPSLAW)*, pages 117–134, London, England, UK, 1995. Springer London.

[SCKK18] Mohammad Shameem, Bibhas Chandra, Rakesh Ranjan Kumar, and Chiranjeev Kumar. A systematic literature review to identify human related challenges in globally distributed agile software development: Towards a hypothetical model for scaling agile methodologies. In *Proceedings of the 4th International Conference on Computing Communication and Automation (ICCCA)*, pages 1–7, New York, NY, USA, 2018. IEEE.

[Sco91] John Scott. *Social Network Analysis: A Handbook*. SAGE, Thousand Oaks, CA, USA, 1991.

[Scr21] ScrumPLoP. Published patterns. `https://sites.google.com/a/scrumplop.org/published-patterns/`, 2021. [Online; accessed 03-04-2022].

[Scr22] Scrum.org. The Scrum Framework Poster. `https://www.scrum.org/resources/scrum-framework-poster`, 2022. [Online; accessed 03-04-2022].

[SH15] Christoph Johann Stettina and Jeannette Hörz. Agile portfolio management: An empirical perspective on the practice in use. *International Journal of Project Management (IJPM)*, 33(1):140–152, 2015.

[SKCK17] Mohammad Shameem, Chiranjeev Kumar, Bibhas Chandra, and Arif Ali Khan. Systematic review of success factors for scaling agile methods in global software development environment: A client-vendor perspective. In *Proceedings of the 24th*

Asia-Pacific Software Engineering Conference Workshops (APSEC), pages 17–24, New York, NY, USA, 2017. IEEE.

[SKCU77] Yutaka Sugimori, Kaneyoshi Kusunoki, Fujio Cho, and Sjtijopr Uchikawa. Toyota production system and kanban system materialization of just-in-time and respect-for-human system. *International Journal of Production Research (IJPR)*, 15(6):553–564, 1977.

[SKL07] Bohdana Sherehiy, Waldemar Karwowski, and John Layer. A review of enterprise agility: Concepts, frameworks, and attributes. *International Journal of Industrial Ergonomics (IJIE)*, 37(5):445–460, 2007.

[SM02] Ken Schwaber and Beedle Mike. *Agile Software Development With Scrum*. Prentice-Hall, Hoboken, NJ, USA, 2002.

[SM15] Alexander Schneider and Florian Matthes. Evolving the eam pattern language. In *Proceedings of the 20th European Conference on Pattern Languages of Programs (EuroPLoP)*, pages 1–11, New York, NY, USA, 2015. ACM.

[SS20] Ken Schwaber and Jeff Sutherland. The scrum guide. `https://scrumguides.org/scrum-guide.html`, 2020. [Online; accessed 03-04-2022].

[Sta95a] Robert Stake. *The Art of Case Study Research*. SAGE, Thousand Oaks, CA, USA, 1995.

[Sta95b] Barry Staw. Repairs on the road to relevance and rigor. In *Publishing in the Organizational Sciences*, volume 2, pages 85–97. SAGE, Thousand Oaks, CA, USA, 1995.

[STE17] Eva-Maria Schön, Jörg Thomaschewski, and María José Escalona. Agile requirements engineering: A systematic literature review. *Computer Standards & Interfaces*, 49:79–91, 2017.

[SVBP07] Jeff Sutherland, Anton Viktorov, Jack Blount, and Nikolai Puntikov. Distributed scrum: Agile project management with outsourced development teams. In *Proceedings of the 40th Hawaii International Conference on System Sciences (HICSS)*, pages 1–10, New York, NY, USA, 2007. IEEE.

[SvECV21] Christoph Johann Stettina, Victor van Els, Job Croonenberg, and Joost Visser. The impact of agile transformations on organizational performance: A survey of teams, programs and portfolios. In *Proceedings of the 22nd International Conference on Agile Software Development (AGILE)*, pages 86–102, Cham, Switzerland, 2021. Springer International.

[TDS03] David Tranfield, David Denyer, and Palminder Smart. Towards a methodology for developing evidence-informed management knowledge by means of systematic review. *British Journal of Management (BJM)*, 14(3):207–222, 2003.

[TFW18] Helena Tendedez, Maria Angela Felicita Cristina Ferrario, and Jonathan Nicholas David Whittle. Software development and cscw: Standardization and flexibility in large-scale agile development. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW):1–23, 2018.

[Tho14] Steven Thompson. *Sampling*. John Wiley & Sons, Hoboken, NJ, USA, 2014.

[TN86] Hirotaka Takeuchi and Ikujiro Nonaka. The new new product development game. *Harvard Business Review (HBR)*, 64(1):137–146, 1986.

[Tor05] Richard Torraco. Writing integrative literature reviews: Guidelines and examples. *Human Resource Development Review (HRDR)*, 4(3):356–367, 2005.

[TSD19] Sven Theobald, Anna Schmitt, and Philipp Diebold. Comparing scaling agile frameworks based on underlying practices. In *Proceedings of the 20th International Conference on Agile Software Development Scientific Workshops (AGILESW)*, pages 88–96, Cham, Switzerland, 2019. Springer International.

[Vai14] Aashish Vaidya. Does dad know best, is it better to do less or just be safe? adapting scaling agile practices into the enterprise. In *Proceedings of the 40th Pacific NW Software Quality Conference (PNSQC)*, pages 828–837, Porland, OR, USA, 2014. Pacific NW Software Quality Conference (PNSQC).

[VBB13] Viswanath Venkatesh, Susan Brown, and Hillol Bala. Bridging the qualitative-quantitative divide: Guidelines for conducting mixed methods research in information systems. *MIS Quarterly (MISQ)*, 37(1):21–54, 2013.

[VBSN+09] Jan Vom Brocke, Alexander Simons, Bjoern Niehaves, Kai Riemer, Ralf Plattfaut, and Anne Cleven. Reconstructing the giant: On the importance of rigour in documenting the literature search process. In *Proceedings of the 17th European Conference on Information Systems (ECIS)*, pages 2206–2217, Atlanta, GA, USA, 2009. AIS.

[vMvV14] Hidde van Manen and Hans van Vliet. Organization-wide agile expansion requires an organization-wide agile mindset. In *Proceedings of the 15th International Conference on Product-Focused Software Process Improvement (PROFES)*, pages 48–62, Cham, Switzerland, 2014. Springer International.

[Wal06] Geoff Walsham. Doing interpretive research. *European Journal of Information Systems (EJIS)*, 15(3):320–330, 2006.

[Wat14] Michael Waterman. *Reconciling Agility and Architecture: A Theory of Agile Architecture*. Dissertation, Victoria University of Wellington, Wellington, New Zealand, 2014.

[WC03] Laurie Williams and Alistair Cockburn. Guest editors' introduction: Agile software development: It? s about feedback and change. *Computer*, 36(6):39–43, 2003.

[WCC12] Xiaofeng Wang, Kieran Conboy, and Oisin Cawley. "leagile" software development: An experience report analysis of the application of lean approaches in agile software development. *Journal of Systems and Software (JSS)*, 85(6):1287–1299, 2012.

[WCP12] Xiaofeng Wang, Kieran Conboy, and Minna Pikkarainen. Assimilation of agile practices in use. *Information Systems Journal (ISJ)*, 22(6):435–455, 2012.

[Wel01] Don Wells. Extreme Programming: A Gentle Introduction. `http://www.extremeprogramming.org/map/project.html`, 2001. [Online; accessed 03-04-2022].

[WJ97] James Womack and Daniel Jones. Lean thinking—banish waste and create wealth in your corporation. *Journal of the Operational Research Society*, 48(11):1144–1150, 1997.

[WJR90] James Womack, Daniel Jones, and Daniel Roos. *The Machine That Changed the World: How Lean Production Revolutionized the Global Car Wars*. Simon and Schuster, New York, NY, USA, 1990.

[WJR07] James Womack, Daniel Jones, and Daniel Roos. *The Machine that Changed the World: The Story of Lean Production–Toyota's Secret Weapon in the Global Car Wars that Is now Revolutionizing World Industry*. Simon and Schuster, New York, NY, USA, 2007.

[WRH+12] Claes Wohlin, Per Runeson, Martin Höst, Magnus Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg, Germany, 2012.

[WW02] Jane Webster and Richard Watson. Analyzing the past to prepare for the future: Writing a literature review. *MIS Quarterly (MISQ)*, 26(2):13–23, 2002.

[WW15] Peter Weill and Stephanie Woerner. Thriving in an increasingly digital ecosystem. *MIT Sloan Management Review (MIT SMR)*, 56(4):27–34, 2015.

[Yıl13] Kaya Yılmaz. Comparison of quantitative and qualitative research traditions: Epistemological, theoretical, and methodological differences. *European Journal of Education (EJE)*, 48(2):311–325, 2013.

[Yin15] Robert Yin. *Case Study Research: Design and Methods*. SAGE, Thousand Oaks, CA, USA, 2015.

[ZC06] Ted Zorn and Nittaya Campbell. Improving the writing of literature reviews through a literature integration exercise. *Business Communication Quarterly (BCQ)*, 69(2):172–183, 2006.

Publications

[PUH+21] Abheeshta Putta, Ömer Uludağ, Shun-Long Hong, Maria Paasivaara, and Casper Lassenius. Why do organizations adopt agile scaling frameworks? a survey of practitioners. In *Proceedings of the 15th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–12, New York, NY, USA, 2021. ACM.

[PUPH21] Abheeshta Putta, Ömer Uludağ, Maria Paasivaara, and Shun-Long Hong. Benefits and challenges of adopting safe-an empirical survey. In *Proceedings of the 22nd International Conference on Agile Software Development*, Cham, Switzerland, 2021. Springer International.

[UHK+18] Ömer Uludağ, Matheus Hauder, Martin Kleehaus, Christina Schimpfle, and Florian Matthes. Supporting large-scale agile development with domain-driven design. In *Proceedings of the 19th International Conference on Agile Software Development (AGILE)*, pages 232–247, Cham, Switzerland, 2018. Springer International.

[UHM19] Ömer Uludağ, Nina-Mareike Harders, and Florian Matthes. Documenting recurring concerns and patterns in large-scale agile development. In *Proceedings of the 24th European Conference on Pattern Languages of Programs (EuroPLoP)*, New York, NY, USA, 2019. ACM. `https://doi.org/10.1145/3361149.3361176`. The published version of this paper is included in Appendix A of the dissertation.

[UKCM18] Ömer Uludağ, Martin Kleehaus, Christoph Caprano, and Florian Matthes. Identifying and structuring challenges in large-scale agile development based on a structured literature review. In *Proceedings of the 22nd International Enterprise Distributed Object Computing Conference (EDOC)*, pages 191–197, New York, NY, USA, 2018. IEEE. `https://doi.org/10.1109/EDOC.2018.00032`. The accepted version of this paper is included in Appendix A of the dissertation.

[UKD+19] Ömer Uludağ, Martin Kleehaus, Niklas Dreymann, Christian Kabelin, and Florian Matthes. Investigating the adoption and application of large-scale scrum at a german automobile manufacturer. In *Proceedings of the 14th International Conference on Global Software Engineering (ICGSE)*, pages 22–29, New York, NY, USA, 2019.

IEEE. https://doi.org/10.1109/ICGSE.2019.00019. The accepted version of this paper is included in Appendix A of the dissertation.

[UKEM19] Ömer Uludağ, Martin Kleehaus, Soner Erçelik, and Florian Matthes. Using social network analysis to investigate the collaboration between architects and agile teams: A case study of a large-scale agile development program in a german consumer electronics company. In *Proceedings of the 20th International Conference on Agile Software Development (AGILE)*, pages 137–153, Cham, Switzerland, 2019. Springer International. https://doi.org/10.1007/978-3-030-19034-7_9. The published version of this paper is included in Appendix A of the dissertation.

[UKRM19] Ömer Uludağ, Martin Kleehaus, Niklas Reiter, and Florian Matthes. What to expect from enterprise architects in large-scale agile development? A multiple-case study. In *Proceedings of the 25th Americas Conference on Information Systems (AMCIS)*, pages 2683 – 2692, Atlanta, GA, USA, 2019. AIS. The published version of this paper is included in Appendix A of the dissertation.

[UKXM17] Ömer Uludağ, Martin Kleehaus, Xian Xu, and Florian Matthes. Investigating the role of architects in scaling agile frameworks. In *Proceedings of the 21st International Enterprise Distributed Object Computing Conference (EDOC)*, pages 123–132, New York, NY, USA, 2017. IEEE. https://doi.org/10.1109/EDOC.2017.25. The accepted version of this paper is included in Appendix A of the dissertation.

[UM19] Ömer Uludağ and Florian Matthes. Identifying and documenting recurring concerns and best practices of agile coaches and scrum masters in large-scale agile development. In *Proceedings of the 26th International Conference on Pattern Languages of Programs (PLoP)*, pages 191–197, Munich, Germany, 2019. The Hillside Group. https://dl.acm.org/doi/10.5555/3492252.3492271. The published version of this paper is included in Appendix A of the dissertation.

[UM20a] Ömer Uludağ and Florian Matthes. Investigating the role of enterprise architects in supporting large-scale agile transformations: A multiple-case study. In *Proceedings of the 26th Americas Conference on Information Systems (AMCIS)*, Atlanta, GA, USA, 2020. AIS. The published version of this paper is included in Appendix A of the dissertation.

[UM20b] Ömer Uludağ and Florian Matthes. Large-scale agile development patterns for enterprise and solution architects. In *Proceedings of the 25th European Conference on Pattern Languages of Programs (EuroPLoP)*, New York, NY, USA, 2020. ACM.

[UNH19] Ömer Uludağ, Sascha Nägele, and Matheus Hauder. Establishing architecture guidelines in large-scale agile development through institutional pressures: A single-case study. In *Proceedings of the 25th Americas Conference on Information Systems (AMCIS)*, pages 551 – 560, Atlanta, GA, USA, 2019. AIS.

[UNHM21] Ömer Uludağ, Sascha Nägele, Matheus Hauder, and Florian Matthes. A tool supporting architecture principles and guidelines in large-scale agile development. In *Architecting the Digital Transformation*, pages 327–344. Springer International, Cham, Switzerland, 2021.

[UPM19] Ömer Uludağ, Henderik A Proper, and Florian Matthes. Investigating the establishment of architecture principles for supporting large-scale agile transformations. In *Proceedings of the 23rd International Enterprise Distributed Object Computing Conference (EDOC)*, pages 41–50, New York, NY, USA, 2019. IEEE.

[UPP+22] Ömer Uludağ, Pascal Philipp, Abheeshta Putta, Maria Paasivaara, Casper Lassenius, and Florian Matthes. Revealing the state of the art of large-scale agile development research: A systematic mapping study. *Journal of Systems and Software (JSS)*, 194:1–43, 2022. `https://doi.org/10.1016/j.jss.2022.111473`. The accepted version of this paper is included in Appendix A of the dissertation.

[UPPM21] Ömer Uludağ, Abheeshta Putta, Maria Paasivaara, and Florian Matthes. Evolution of the agile scaling frameworks. In *Proceedings of the 22nd International Conference on Agile Software Development (AGILE)*, pages 123–139, Cham, Switzerland, 2021. Springer International. `https://doi.org/10.1007/978-3-030-78098-2_8`. The published version of this paper is included in Appendix A of the dissertation.

[URM21] Ömer Uludağ, Niklas Reiter, and Florian Matthes. Improving the collaboration between enterprise architects and agile teams: A multiple-case study. In *Architecting the Digital Transformation*, pages 347–366. Springer International, Cham, Switzerland, 2021.

Abbreviations

AGILE International Conference on Agile Software Development

AMCIS Americas Conference on Information Systems

ART Agile Release Train

CoP Community of Practice

DAD Disciplined Agile Delivery

EAM Enterprise Architecture Management

EDOC International Enterprise Distributed Object Computing Conference

EPLoP European Conference on Pattern Languages of Programs

ICGSE International Conference on Global Software Engineering

IT Information Technology

HSD Holistic Software Development

JSS Journal of Systems and Software

LeSS Large-Scale Scrum

MIT Massachusetts Institute of Technology

NPS Net Promoter Score

P Publication

Parallel Parallel Agile

PDR Pattern-based Design Research

PI Program Increment

PLoP International Conference on Pattern Languages of Programs

RAGE Recipes for Agile Governance in the Enterprise

RTE Release Train Engineer

SAFe Scaled Agile Framework

sebis Chair for Software Engineering for Business Information System

SoS Scrum-of-Scrums

Spotify Spotify Model

TPS Toyota Production System

TUM Technical University of Munich

WIP Work-in-Progress

XP Extreme Programming

APPENDIX A

Embedded Publications in Original Format

Revealing the State of the Art of Large-Scale Agile Development Research: A Systematic Mapping Study

Ömer Uludağ[a,*], Pascal Philipp[a], Abheeshta Putta[b], Maria Paasivaara[b,c], Casper Lassenius[b,d], Florian Matthes[a]

[a]*Technical University of Munich, Department of Informatics, Munich, Germany*
[b]*Aalto University, Department of Computer Science, Espoo, Finland*
[c]*LUT University, Finland*
[d]*Simula Research Laboratory, Oslo, Norway*

Abstract

Context: Success with agile methods in the small scale has led to an increasing adoption also in large development undertakings and organizations. Recent years have also seen an increasing amount of primary research on the topic, as well as a number of systematic literature reviews. However, there is no systematic overview of the whole research field. *Objective:* This work identifies, classifies, and evaluates the state of the art of research in large-scale agile development.
Method: We conducted a systematic mapping study and rigorously selected 136 studies. We designed a classification framework and extracted key information from the studies. We synthesized the obtained data and created an overview of the state of the art.
Results: This work contributes with (i) a description of large-scale agile endeavors reported in the industry, (ii) a systematic map of existing research in the field, (iii) an overview of influential studies, (iv) an overview of the central research themes, and (v) a research agenda for future research.
Conclusion: This study portrays the state of the art in large-scale agile development and offers researchers and practitioners a reflection of the past thirteen years of research and practice on the large-scale application of agile methods.

Keywords: Agile software development, large-scale agile development, systematic mapping study

1. Introduction

Contemporary business environments are characterized by high unpredictability due to rapidly shifting customer demands and technological advancements, implying that flexibility, adaptability, and learning are crucial to business success [76]. Over the past decades, software has become an integral part of many products and services [81]. To react quickly to changing environments and fluctuating customer requirements, the agile movement emerged in the 1990s, leading to the creation of the Agile Manifesto[1] and many agile methods, e.g., Extreme Programming (XP) [14] and Scrum [94] [54, 81]. Agile methods were originally designed for small, co-located, and self-organizing teams that produce software in close collaboration with business customers, using regular feedback and rapid development iterations [32, 36]. The successful application of agile methods in small projects inspired companies to increasingly adopt agile methods also in large-scale projects and organizations [35]. During the last decade, agile methods have been extended to better fit large-scale settings. Several scaling frameworks have been created both by some custodians of existing agile methods and by others who have worked with companies in scaling agile methods to their settings [101]. As the frameworks claim to provide off-the-shelf solutions to the problems of scaling, their adoption has rapidly increased in practice, as confirmed by the latest State of Agile survey [31]. The survey shows that many large software-intensive companies have adopted scaling frameworks to address challenges accompanied by the scaling of agile methods [5, 24]. While there are more than 20 available frameworks [101], the most popular ones are [31]: Scaled Agile Framework (SAFe) [51], Large-Scale Scrum (LeSS) [23], and Disciplined Agile Delivery (DAD) [8].

As the popularity of applying agile methods at scale has increased in the industry, scientific research on the topic has emerged in recent years. Eleven years ago, at the International Conference on Agile Software Development, industrial practitioners were asked to create a backlog of topics they think should be studied. They voted "*Agile and large projects*" as a top burning research question [41]. After that, nine International Workshops on Large-Scale Agile Development at the yearly International Conferences on Agile Software Development have gathered researchers and practitioners to discuss recent large-scale agile development studies and create research agendas for future research on that area (cf. [33, 70]). In recent years, research publications on large-scale agile have been published at scientific conferences and journals, and the body of knowledge has grown immensely (cf. [32, 40]). Although scientific articles on large-scale agile development are mainly primary studies, we also identified several secondary studies that synthesize the scientific knowledge on large-scale agile devel-

*Corresponding author
Email address: oemer.uludag@tum.de (Ömer Uludağ)
[1] https://agilemanifesto.org/, last accessed on: 19-04-2022.

opment related to specific topics, e.g., the systematic literature review on "*challenges and success factors for large-scale agile transformations*" [32]. A steadily growing number of primary studies is a valuable indicator of the increasing maturity of a research field. It leads to a critical tipping point when secondary studies can be conducted, which then facilitates the aggregation of the results of the primary studies. As the research area of large-scale agile development matures and the number of studies increases, there is a need to systematically identify, analyze, and classify the state of the art of this research field. While the number of primary studies is sufficient, so far, no systematic mapping study has been published to provide a comprehensive overview of the state of research in this area. This mapping study aims to close this gap and provide an overview of the research activities pertaining to large-scale agile development. The main contributions of this study are as follows:

- A *description* of large-scale agile endeavors and their characteristics reported in the industry,
- A *systematic map* classifying, comparing, and evaluating existing research on large-scale agile development,
- An *overview* of the most influential studies on large-scale agile development,
- An *overview* of the central research themes and main findings on large-scale agile development, and
- A *research agenda* consisting of a detailed description of research gaps and a list of open research questions in large-scale agile development.

The remainder of this paper is structured as follows. Section 2 presents background and related work. Section 3 portrays the procedure of the systematic mapping study. Section 4 shows the results of this study. Section 5 discusses the results. Section 6 describes the threats to validity. Section 7 concludes the paper along with remarks on future research.

2. Background and related work

This section provides the background on agile software development and large-scale agile development and gives an overview of related work, including other secondary studies.

2.1. Agile software development

Software development methods have undergone a dramatic evolution from traditional sequential development approaches to more flexible and adaptive development approaches due to the market's increasing demands and customer requirements volatility [81]. To address these needs, many companies have started to adopt agile software development methods [80]. Various software practitioners introduced several agile methods in recent years, e.g., the Crystal Method, the Dynamic System Development Method, and the Feature-Driven Development Method, that comprise a set of iterative and incremental methods based on specific values and principles defined in the Agile Manifesto [1]. Currently, the most widely adopted agile methods in industry are Scrum and XP [44]. These methods have aroused great interest both in practice and academia [3].

Agile methods have received both acceptance and criticism in the industry [20]. On the one hand, they have proven to be successful in improving quality [93], productivity [39], and customer satisfaction [39]. On the other hand, there is concern that these methods may not be suitable for large-scale environments [39]. Dybå and Dingsøyr [39] note that there is evidence that suggests combining agile and traditional methods, i.e., hybrid methods, in large undertakings and recommends that practitioners should carefully examine and compare project characteristics with the required characteristics of the suitable agile method(s). Previous literature has also reported on the success of hybrid approaches [67], e.g., recent research by Klünder et al. [59] concludes that projects devising hybrid methods have about a 5% better chance of achieving their goals.

According to Digital.ai [31], companies adopt agile methods to accelerate software delivery, manage changing priorities, increase productivity, improve alignment between business and IT, and enhance quality, to name a few. However, adopting agile methods is not easy, as agile methods do not rely solely on the appropriate application of individual tools or practices but rather often demand a holistic way of thinking and mindset. Thus, the adoption of agile methods often requires a change in the entire organizational culture [69]. Developing an agile mindset and changing the company's culture takes time and effort, and if this effort is neglected, the organization can fall back into old habits and fail to reap the full benefits of agility [61].

2.2. Large-scale agile development

The success of agile methods for small, co-located teams has incited companies to increasingly apply agile practices to large-scale projects [32, 36, 84]. However, the large-scale adoption of agile methods has proven to be very challenging [38]. The challenges of adopting agile practices at large-scale are partly related to the organization's size, as the difficulties of adopting agile methods increase with the size of the organization, i.e., in large organizations, products are more complex, and the inter-dependencies between teams are greater than in smaller organizations [7, 38], leading to inertia and slowing down the change process [66]. Another challenge in large organizations is the need for coordination and communication between multiple teams and also between different organizational units that often do not work in an agile manner, requiring additional coordination mechanisms between teams and also organizational units [65]. While agile methods primarily focus on intra-team practices that work well in small organizations, agile methods do not provide sufficient guidance on how agile teams should interact in large environments [68]. Hence, large organizations must adapt the practices to their specific needs. As a result, practices may need to be put in place that require additional formal communication, which might reduce their agility [65]. As often large organizations are globally distributed and agile methods are primarily based on frequent internal and external collaboration and communication [49], the use of agile practices in globally distributed projects can be challenging [46].

To address issues associated with adopting agile practices in large-scale organizations and projects, consultants and software practitioners have proposed several scaling agile frameworks,

e.g., SAFe, LeSS, and DAD, which include predefined workflow patterns to deal with concerns related to large numbers of teams, inter-team coordination, and customer involvement [5, 36]. As large organizations are increasingly pressured and expected to become more agile, and scaling frameworks claim to provide off-the-shelf solutions to scaling, companies have begun to adopt these frameworks at an increasing rate in recent years [24]. This trend is also confirmed by the annual non-scientific survey on the State of Agile by Digital.ai [31].

2.2.1. Definition of large-scale agile development

"*When can a company be said to be adopting agile methods at scale rather than just at a smaller scale?*" Several researchers have already tried to answer this question and have proposed several definitions on what "*large-scale agile development*" means. These definitions usually include the number of people or agile teams engaged in the effort or associated costs or duration of a project [35]. Berger and Beynon-Davies [18], for example, classify a project as a large-scale agile project if the project costs exceed 10 million GBP. Bjarnason et al. [19] use the project duration of more than two years as an indicator for classifying a project as large-scale. According to Paasivaara et al. [78], a project with more than 40 people or seven agile teams involved can be considered large-scale. To bring a conceptual clarity of what "*large-scale agile development*" means, Dingsøyr et al. [35] have identified several different interpretations and proposed a taxonomy for large-scale agile development that uses the number of collaborating and coordinating teams to define the scale of an agile project. The taxonomy developed by Dingsøyr et al. [35] consists of three categories: (i) small-scale agile projects with one team that can use traditional agile practices, e.g., daily meetings, sprint planning, review, and retrospective meetings, for intra-team coordination, (ii) large-scale agile projects with 2–9 agile teams that use new forums, e.g., a Scrum-of-Scrums (SoS) for cross-team coordination, and (iii) very large-scale agile projects with at least 10 agile teams that require several forums for inter-team coordination, e.g., multiple SoS. According to Dingsøyr et al. [35], a project can be considered a large-scale project if it has at least two coordinating agile teams. Fuchs and Hess [42] extend this definition and state that the term "*large-scale agile development*" has multiple interpretations: (i) the use of agile methods in large teams, (ii) the employment of agile methods in large organizations, (iii) the application of agile methods in large multi-team settings, i.e., "*large agile multi-team settings*", or (iv) the usage of agile practices in organizations as a whole, i.e., "*organizational agility*"[2]. Like Fuchs and Hess [42], we focus on the latter two definitions and understand the large-scale agile adoption of agile methods in large agile multi-team settings with at least two teams or the large-scale adoption of agile methods on the organizational level comprising multiple large agile multi-team settings.

[2]The term "*organizational agility*" should not be confused with the term "*enterprise agility*", which constitutes a research direction for itself. According to our understanding, the term "*enterprise agility*" comprises the adoption of agile methods at the company level. In contrast, the term "*organizational agility*" insinuates the adoption of agile methods in large organizational units of companies, e.g., departments, divisions, or units. Hence, the term "*organizational agility*" can be seen as a subset of the term "*enterprise agility*".

2.3. Secondary studies on large-scale agile development

Several secondary studies have systematically analyzed the literature on specific topics related to large-scale agile development. We identified 13 secondary studies (see Table 1), of which 10 were systematic literature reviews, and three were either structured or simple literature reviews. Systematic literature reviews have addressed several topics, e.g., the identification and description of challenges (cf. [32, 95, 100]), success factors (cf. [32, 95]), and typical roles involved in large-scale agile endeavors (cf. [43, 99]). Various researchers also conducted systematic literature reviews and multi-vocal literature reviews on scaling frameworks to identify challenges, benefits, and success factors related to the adoption of scaling frameworks (cf. [40, 53, 84]). We also identified simple literature reviews comparing various scaling frameworks (cf. [5, 77]).

Existing secondary studies cover and analyze specific research topics on large-scale agile development in-depth, e.g., large-scale agile transformations, scaling agile frameworks, or global and distributed software engineering. For instance, Dikert et al. [32] deal with the research topic of large-scale agile transformations and explore the challenges and success factors reported for these transformations. For example, Edison et al. [40] address the research topic of scaling agile frameworks by analyzing and comparing them based on their principles, practices, tools, and metrics, as well as identify gaps in the literature and proposing needs for future research. Shameem et al. [95] deal with the research topic of global and distributed software engineering by identifying key success factors for scaling agile methods in these environments.

While existing studies focus on specific research topics in large-scale agile development and analyze them in-depth, we could not identify any systematic mapping studies that would provide an overview of the overall state of research in the field, structuring the body of knowledge in large-scale agile development. Hence, this paper aims to fill this gap by providing an overview of the research activities in large-scale agile development based on a systematic mapping study. As a systematic mapping study, the scope is broader than any existing systematic literature review, and the goals differ. While there is some overlap between this study and the systematic literature reviews of Dikert et al. [32] and Edison et al. [40], we provide a broader overview of the overall field of large-scale agile development than any single systematic literature review.

3. Research process

While systematic literature reviews [21, 56] are a common means for identifying, evaluating, interpreting, and comparing all available research related to a particular research question, a systematic mapping study maps out the existing research rather than answering a detailed research question [22, 82]. Hence, we opted to conduct a systematic mapping study, as it is capable of dealing with broad research areas and provides a systematic and

Table 1: Secondary studies on large-scale agile development

Year	Study	No. of studies	Topic
2014	[86]	75	This study provides an understanding of research problems and themes in large-scale, distributed agile development environments based on IEEE publications between 2005 and 2014.
2015	[92]	51	This paper identifies research issues related to the scalability of agile methods for large-scale projects. Moreover, this study unveils existing methods, approaches, frameworks, and practices that can facilitate the application of agile methods for large-scale agile projects, as well as their limitations.
2016	[32]	52	This article answers the question of how large organizations or projects adopt agile and/or lean methods at scale by focusing on the reported challenges and success factors encountered in large-scale agile transformations.
2017	[43]	42	This work provides an analysis of roles assigned with the responsibility for inter-team coordination of large-scale agile development settings.
2017	[6]	60	This paper presents challenges when developing quality requirements in large-scale distributed agile projects and reveals agile practices that have contributed to the emergence of these challenges. This work also summarizes solutions proposed in the literature to address challenges associated with developing quality requirements in large-scale distributed agile projects.
2017	[95]	20	This study presents key factors that can positively impact agile development activities in large-scale, globally distributed software development environments.
2017	[99]	146	This paper gives an overview of 20 identified scaling agile frameworks and describes the responsibilities of different architects in these frameworks.
2018	[100]	76	This work presents different stakeholders and their recurring challenges in large-scale agile projects.
2018	[96]	18	This article shows an overview of human-related factors that can negatively impact agile practices in large-scale, globally distributed software development environments and proposes a hypothetical model of the identified challenges related to the scaling of agile methods.
2018	[84]	88	This study provides an analysis of the scientific and grey literature describing the challenges and benefits encountered by organizations when adopting SAFe.
2018	[53]	12	This work examines practices, challenges, and success factors for scaling agile methods in large companies, reported in the literature and within a large software company.
2019	[4]	58	This paper shows a set of motivators for the large-scale adoption of agile methods from a management perspective.
2021	[40]	191	This article compares five scaling methods based on each method's principles, practices, tools, and metrics. It also presents the challenges and success factors described in the literature when applying these methods.

objective procedure for identifying, categorizing, and analyzing the existing literature [22, 56, 82].

3.1. Objectives and research questions

We used the Goal-Question-Metric paradigm [11] to formulate the objective of this study: to **analyze** peer-reviewed literature **for the purpose of** providing an overview of the state of the art **with respect to** the characterization of the topic, the available research on the topic, salient publications and researchers, well-established research streams, and potential research gaps **from the point of view of** scholars and practitioners **in the context of** large-scale agile development. The overall research question of this study is:

Research Question: *What is the state of the art of the literature pertaining to large-scale agile development?*

To answer this question, we further decomposed it into four specific questions:

RQ1: *How is large-scale agile development characterized in the literature?*

Currently, there is no consensus on the actual meaning of the term "*large-scale agile development*" [90] which is why the lack of conceptual clarity regarding this term inhibits effective collaboration and progress in the research area of large-scale agile development [35]. Thus, this research question strives to explore how the extant literature characterizes this term based on the reported case characteristics.

RQ2: *What are the publication trends and characteristics of existing research on large-scale agile development?*

A valuable instrument for understanding the nature of a research area is the investigation of research trends and the systematic classification of extant studies [22, 82]. Accordingly, this research question intends to map the frequency of publications over time to identify research trends and strives to categorize and aggregate extant studies to structure the research area.

RQ3: *What are the seminal studies in large-scale agile development?*

There is perhaps no better way to understand and explore the intellectual structure of a research field than to identify its seminal works [34, 71]. Therefore, this research question aims to identify the protagonists and salient publications in the research field by employing bibliometric analysis.

RQ4: *Which research streams and promising future research directions exist in large-scale agile development?*

One approach to assess the state of the art and maturity level of a research area is to identify main research streams and reveal potential research gaps [58, 82]. Hence, this research question strives to map the general structure of the research field by identifying central research themes. This research question also aims to outline a research agenda for future research efforts by analyzing existing gaps in the research streams.

3.2. Mapping study execution

When conducting this study, we followed the guidelines for performing systematic mapping studies [82] and systematic literature reviews [56]. We decided to combine both approaches [55] since two of our research questions, namely *RQ1* and *RQ4*, could not be answered by mappings alone. The execution procedure of this systematic mapping study consisted of three phases: (i) study search, (ii) study selection, (iii) and data extraction, as described in the following.

3.2.1. Study search

In the first phase, we conducted a two-step study search procedure, which included the definition of a search strategy and the screening of related studies consisting of a preliminary search and main search, as described subsequently.

Study search strategy. Defining a proper search strategy is essential to ensure that the literature review results are complete [105]. Various researchers have proposed several techniques to develop appropriate search strategies (cf. [83, 105]). We followed the recommendations by Zhang et al. [105] to elaborate on our search strategy, which we describe in the following.
Search approach. Our search comprised two main steps: preliminary search and main search. The purpose of the preliminary search was two-fold. First, we wanted to use the preliminary search to construct and evaluate different search strings for the main search. Second, we used the preliminary search as a "*sanity check*" to identify a set of relevant papers that the actual main search should also retrieve. Following the preliminary search, we performed a database keyword search during the main search to retrieve relevant studies in electronic databases listed in Table 2. Afterward, we merged the search results from the preliminary and main searches and excluded duplicate studies. We then included the resulting collection of potentially relevant papers for the study selection phase.

Table 2: Databases used in the main search

#	Search engine	Website
DB1	IEEE Xplore	`http://ieeexplore.ieee.org/`
DB2	ACM Digital Library	`http://dl.acm.org/`
DB3	Science Direct	`http://www.sciencedirect.com/`
DB4	Web of Science	`https://www.webofknowledge.com/`
DB5	Scopus	`https://www.scopus.com/home.uri`
DB6	AIS eLibrary	`https://aisel.aisnet.org/`

Data sources. According to Brereton et al. [21], many different electronic sources should be searched since no single source can find all relevant primary studies. Therefore, as suggested by Kitchenham and Brereton [55], we selected six electronic databases (see Table 2) as the primary sources for the systematic mapping study for covering as many potentially relevant studies as possible. The selection of the electronic databases was guided by: (i) the fact that two of them, i.e., ACM Digital Library and IEEE Xplore, are the largest scientific databases in the field of software engineering [55, 83], (ii) the fact that three of them offer broad coverage of diverse research fields, i.e., Science Direct, Web of Science, and Scopus [89], (iii) and the fact that one of them, i.e., AIS eLibrary, contains articles from the primary information systems research dissemination outlets [74]. We excluded Google Scholar as the results tend to overlap with ones from the included electronic databases [25].
Search terms. To identify all relevant studies, we used a five-step strategy [57] for constructing the search terms:

1. deriving main search terms from the study topic and the formulated research questions based on the PICO (Population, Intervention, Comparison, Outcomes) criteria,
2. identifying synonyms and alternative spellings for the main search terms,
3. checking the keywords in relevant papers,
4. incorporating synonyms and alternative words using the Boolean *OR* operator, and
5. linking the search terms using the Boolean *AND* operator.

We only used the first two components of the PICO approach, i.e., population and intervention, and omitted the outcome and context facets from the search structure since our research questions did not warrant a restriction of the results to a specific outcome or context. Similar to Yang et al. [104], the population facet represents the first search set of the overall search string and contains the terms of agile methods that are popularly used in various systematic literature reviews and surveys on agile software development (cf. [38, 88, 98]). Following Dikert et al. [32], we extended the first search set by explicitly stating that the application of agile methods outside of software engineering, e.g., agile manufacturing, should be excluded. The intervention comprises two search sets. The first set includes terms related to the objective of applying agile methods on a larger scale, namely "*large-scale*" and "*scaling*". These two terms are often used within titles and as keywords in related publications on large-scale agile development (cf. [36, 53]). Inspired by Yang et al. [104], the second intervention search set entails terms of large-scale agile development methods and frameworks. We used the results of Uludağ et al. [99] to obtain a list of these methods and frameworks. Following this strategy, we conducted a series of tests and refinements in the preliminary search. The blending of the search sets resulted in the generic search string for the main search. The final generic search string used was:

Agile software development ***AND*** *(Large-scale development* ***OR*** *Scaling agile frameworks)*

Table 3 lists the final list of applied search sets and strings. As each electronic database has a specific syntax for search terms, we adapted our search string to the particular syntax requirements of the search engines.
Time span. We cover the period from February 2001, when the

Table 3: Overview of search sets and corresponding terms

Set	Search term
Agile software development	((agile **OR** agility **OR** extreme programming **OR** XP **OR** feature driven development **OR** FDD **OR** scrum **OR** crystal **OR** pair programming **OR** test-driven development **OR** TDD **OR** leanness **OR** lean software development **OR** lean development **OR** LSD) **AND NOT** manufacturing)
	AND
Large-scale development	((large-scale **OR** scaling)
	OR
Scaling agile frameworks	(Crystal Family **OR** Dynamic Systems Development Method Agile Project Framework for Scrum **OR** Scrum of Scrums **OR** Enterprise Scrum **OR** Agile Software Solution Framework **OR** Large-Scale Scrum **OR** Scaled Agile Framework **OR** Disciplined Agile **OR** Spotify Model **OR** Mega Framework **OR** Enterprise Agile Delivery and Agile Governance Practice **OR** Recipes for Agile Governance in the Enterprise **OR** Continuous Agile Framework **OR** Scrum at Scale **OR** Enterprise Transition Framework **OR** ScALeD Agile Lean Development **OR** eXponential Simple Continuous Autonomous Learning Ecosystem **OR** Lean Enterprise Agile Framework **OR** Nexus **OR** FAST Agile))

Agile Manifesto was proposed, to the end of December 2019, when we started this systematic mapping study.

Preliminary and main search. Figure 1 shows the study search process and the individual results obtained in each of both phases of the study search. In the preliminary search, we retrieved 693 studies. After removing duplicate studies, 631 papers were left. The main search returned 2,090 publications. After removing duplicate papers, we ended up with 1,643 papers from the main search. After merging the search results from the preliminary and main search and removing duplicates, we retrieved a total of 2,144 articles that serve as input for the subsequent study selection process (see Section 3.2.2).

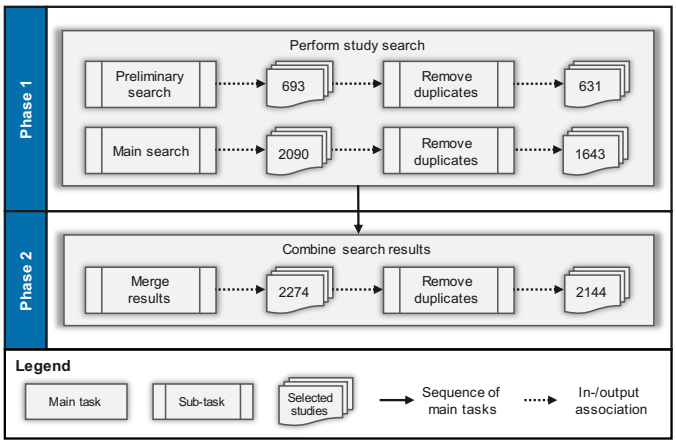


Figure 1: Overview of the study search process

3.2.2. Study selection

After the study search process, we considered all 2,144 studies for the subsequent study selection consisting of three screening phases: (i) selection of relevant articles based on their metadata (incl. title, keywords, publication year, and publication type), (ii) selection of relevant studies based on their abstract, and (iii) selection of relevant papers based on their full-text. The study selection was based on explicit inclusion and exclusion criteria and was conducted by two researchers in parallel. Following the individual decisions, the researchers harmonized their selection results and resolved conflicts. To ensure that the study selection results were objective, we created a set of well-defined inclusion and exclusion criteria employed in the selection process to filter relevant articles for our study.

Selection criteria. We defined selection criteria to reduce the likelihood of bias and to assess the relevance of the studies [56]. Before the study selection process, two researchers discussed and reached an understanding of the inclusion and exclusion criteria (see Table 4). We included an article if it satisfied *all* specified inclusion criteria and discarded it if it met *any* exclusion criterion. We only included peer-reviewed papers (see I3) that describe completed research results (see I4) and cover the large-scale application of agile methods (see I1 and I2), i.e., the application of agile methods in large multi-team settings or the usage of agile practices in organizations as a whole (see Section 2.2.1). Besides articles meeting any of the exclusion criteria (see E1 – E7), we also excluded papers that did not indicate large-scale considerations or described the single team adoption of agile methods, not fulfilling I2.

Table 4: Inclusion and exclusion criteria

ID	Criteria	Assessment criteria
I1	Inclusion	Describe the application of agile methods in software development.
I2	Inclusion	Cover the application of agile methods on a large scale and meet the requirements of being large-scale based on our understanding and definition of large-scale agile development in Section 2.2.
I3	Inclusion	Peer-reviewed, i.e., published in journals, conference or workshop proceedings.
I4	Inclusion	Papers that describe completed research results.
E1	Exclusion	Related to agile manufacturing.
E2	Exclusion	Published in the form of abstracts, book chapters, book and conference reviews, grey literature, magazines, newsletter, short communications, talks, technical reports, and tutorials.
E3	Exclusion	Articles that are not written in English language.
E4	Exclusion	Published before the creation of the Agile Manifesto.
E5	Exclusion	Not available as a full-text.
E6	Exclusion	Experience reports and opinion papers.
E7	Exclusion	Previous version(s) of extended papers.

Selection process. The selection process consisted of three phases (see Figure 2). By the end of the third phase, the first researcher marked 269 studies as relevant, while the second researcher marked 145 studies as related. By the end of the fourth phase, 136 studies were characterized by both researchers as pertinent after resolving conflicts (inclusion rate of 6.34%).

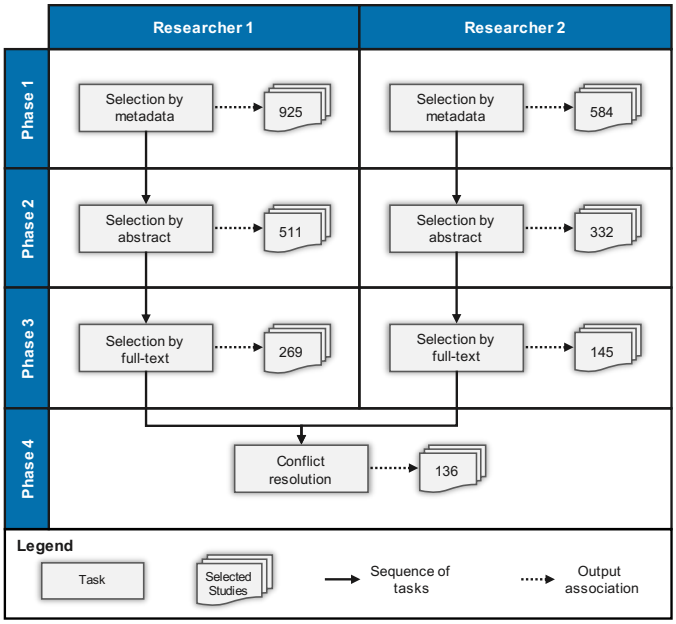


Figure 2: Overview of the study selection process

3.2.3. Data extraction

To facilitate data extraction and simplify management of the extracted data, we adopted the approach of categorizing studies into facets [82] and designed a rigorous classification framework based on these facets. Similar to study selection, we used a spreadsheet to record the extracted data. To reduce bias in the data extraction results, two researchers performed the data extraction independently. Before the formal data extraction process, two researchers discussed the definitions of the data items to be extracted to ensure that both researchers had a common understanding. After both scientists completed the data extraction, they discussed their results together and resolved conflicts to reach a consensus on the results. Figure 3 shows the resulting classification framework, which consists of four facets and several subordinate data items.

Characterization of large-scale agile development (RQ1). To compile information about the characterization of large-scale agile development, we collected information from the reported case characteristics regarding the four data items: *large-scale agile development category*, *case company characteristics*, *organizational agility characteristics*, and *large agile multi-team setting characteristics*. We extracted data for the *large-scale agile development category* data item to bring more conceptual clarity regarding the actual meaning of the term "*large-scale agile development*". To this end, we used the classification by Fuchs and Hess [42] to categorize the large-scale agile efforts of the reported companies. To get a better picture of companies adopting agile methods at scale, we divided the *case company characteristics* data item into four sub-data items, namely *company name*, *location of company headquarter*, *sector*, and *company size*. To obtain information on the four sub-data items, we combed through the case descriptions. In cases where the name of the company was provided but no other information about the company's location, sector, or size was available, we used the information on the company's website to complete the data. To classify the extracted data related to the *sector* sub-data item, we used the leading Global Industry Classification Standard[3] from MSCI. We used the categorization of Digital.ai [31] for classifying the reported case companies according to their *company size*. Using this categorization also helped us to compare our results more easily with those of Digital.ai [31]. To understand the trend of organizational agility and compile information related to the *organizational agility characteristics* data item, we read the case descriptions of the selected papers to determine the start date of the large-scale agile transformations of the reported case companies. To characterize the reported large agile multi-team settings, we split the *large agile multi-team setting characteristics* data item into three sub-data items, namely *scaling and complexity factors*, *organizational size*, and *applied development and scaling approaches*. We used an integrated approach [28] to extract data related to the *scaling and complexity factors* sub-data item. In doing so, we used the scaling and complexity factors reported by Ambler [7] as a starting list and iteratively expanded it as we identified new factors. For the *organizational size* sub-data item, we read the case descriptions and looked for information on the number of agile teams and people involved in the agile multi-team settings. We used the taxonomy of Dingsøyr et al. [35] to determine whether a setting was small-scale, large-scale, or very large-scale. To extract data related to the *applied development and scaling approaches* sub-data item, we applied an integrated approach [28] and used the lists of Abrahamsson et al. [3] and Uludağ et al. [99] for agile development and scaling approaches and refined the initial lists as new approaches were identified in the studies.

Publication trends and research characteristics (RQ2). Figure 3 shows the data items we used to collect data about publication trends and research characteristics. The *publication year*, *country of authorship*, and *publication venue* were retrieved directly from the studies' metadata. Based on the *publication venue*, we derived the *publication type* with a classification scheme consisting of the three categories: *journal*, *conference*, and *workshop*. To classify the *publication domain*, we read the website information of the publication venues and categorized them into one of eight primary publication domains: *enterprise computing*, *information systems*, *IT project management*, *human computer interaction*, *marketing*, *multidisciplinary*, *project management*, and *software engineering*. To classify the research types, we adopted a framework consisting of six categories based on Wieringa et al. [103] (see Table 5). For the *research approach* data item, we used a combination of three taxonomies and definitions of Berg et al. [17], Rodríguez et al. [89], and Unterkalmsteiner et al. [102] to have a complete list of research methods consisting of nine categories (see Table 6). For the *contribution type* data item, we used two existing taxonomies of Shaw [97] and Paternoster et al. [79] to have a complete list of research outcomes consisting of seven categories (see Table

[3] `https://www.msci.com/our-solutions/indexes/gics`, last accessed on: 11-13-2021.

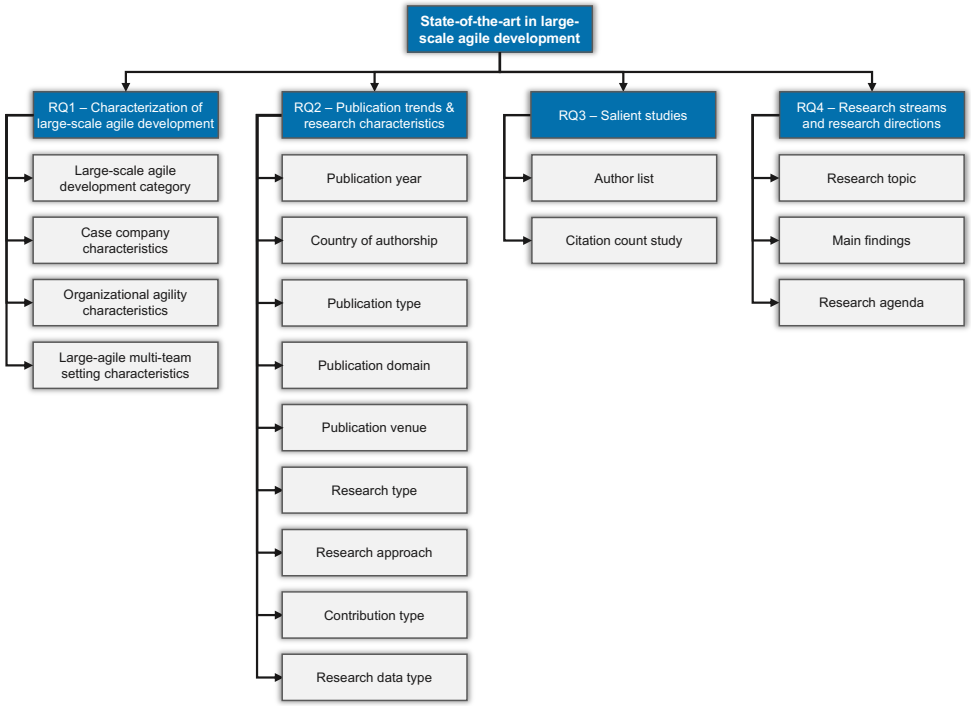


Figure 3: Classification framework

7). The *research data type* consists of three categories indicating whether a study is *primary*, *secondary*, or *tertiary*.

Table 5: Classification scheme for research types based on Wieringa et al. [103]

Research type	Description
Evaluation research	The implementation of existing techniques and solutions are evaluated in practice.
Experience reports	Practitioners report their own experiences from one or more real-life projects without discussing the article's underlying research method.
Opinion papers	Express the author's personal experience regarding a technique's suitability without relying on related work and research methods.
Philosophical papers	These articles sketch a new perspective on looking at existing things by structuring the field using a taxonomy or conceptual framework.
Solution proposal	A new or significant extension of an existing technique is shown by demonstrating its benefits and applicability.
Validation research	Focus on the investigation of novel techniques that have not yet been implemented in practice.

Seminal studies (RQ2). To compile information about influential studies, we used two data items: *author list* and *citation count study*. We obtained the *author list* from the metadata of the selected papers. We collected data for the *citation count study* data item via Google Scholar.

Research streams and research directions (RQ4). To identify central research themes and future research directions, we collected data on research topics and agendas in the selected papers. We followed a systematic process called *keywording* [82] to define the categories of the *research topic* facet. The purpose

Table 6: Classification scheme for research approaches based on Berg et al. [17], Rodríguez et al. [89], and Unterkalmsteiner et al. [102]

Research approach	Description
Action research	Applies action research to solve a real-world problem while simultaneously scrutinizing the experience of solving the problem.
Case study	Uses a case study to provide an in-depth understanding of a real-life situation and contemporary phenomenon or evaluates a theoretical concept by empirically implementing it in a case study.
Design and creation	Creates a new IT product, artifact, model, or method.
Grounded theory	Uses a systematic process to generate theory from the data obtained based on grounded theory.
Mixed methods	Applies more than one research methodology.
Systematic literature review/Systematic mapping study	Collects and analyzes primary data to address a specific research question or topic using a systematic literature review or systematic mapping study.
Survey	Collects quantitative and/or qualitative data in a standardized, systematic way to find patterns through a questionnaire or interviews.
Theoretical	A theoretical study not mentioning grounded theory as the research methodology.
Not stated	Does not state the research method, nor can it be derived or interpreted by reading the paper.

of the keywording process is to effectively develop a classification framework that fits the selected studies and takes their research focus into account [82]. The keywording process consists of the following three steps:

1. *Identifying keywords and concepts:* Two researchers collected keywords and concepts by reading the full-text of each starting study.
2. *Clustering keywords and concepts:* Two researchers per-

Table 7: Classification scheme for contribution types based on Shaw [97] and Paternoster et al. [79]

Contribution type	Description
Advice/Implication	Discursive and general recommendation based on personal opinions.
Framework/Method	Framework or method to facilitate the construction and management of software-intensive systems.
Guideline	List of advice or recommendations based on the synthesis of the research results obtained.
Lessons learned	Set of outcomes which are analyzed from the research results obtained.
Model	Representation of an observed reality using concepts resulting from a conceptualization process.
Theory	Construct of cause-effect relationships from determined results.
Tool	Technology, program, or application developed to support various aspects of software engineering.

formed a clustering operation on the collected keywords and concepts into a set of categories and sub-categories.

3. *Refining classification:* Four researchers discussed the preliminary categories and sub-categories. This discussion resulted in the refinement of the classifications to fit them better with the selected studies.

The above-described process ended when no study was left to analyze. During the extraction process, some studies could be classified into more than one research topic. We read the results, discussion, and conclusion sections of the studies to identify their *main findings* and mapped them to the main research topic categories. We used a deductive approach [28] to categorize the *research agenda* of the selected studies based on the final classification of the *research topic* facet. Two researchers read the selected studies' future work sections and mapped the corresponding text fragments to the identified main research topic categories. In addition to the selected studies, we read and mapped relevant data from nine related workshop summaries. They provide a list of important future research topics proposed by researchers and practitioners familiar with large-scale agile development. Following the coding procedure, two researchers merged and aggregated related codes and reformulated the final codes as research questions.

4. Study results

In this section, we provide a review of the state of the art of large-scale agile development research and present our answers to the formulated research questions (see Section 3.1). This section is arranged according to the research questions.

4.1. Characterization of large-scale agile development

Although we discussed the term "*large-scale agile development*" in Section 2.2.1 and stated our understanding of it, we were curious to learn what other authors refer to as large-scale agile development. In what follows, we provide an overview of case companies we studied and identify which category of large-scale agile development, i.e., large agile multi-team settings or organizational agility, was reported. We then highlight our key findings for each category.

4.1.1. General overview of the case companies

We identified 158 case companies, of which 137 remained unnamed in 67 studies[4]. A total of 21 companies were explicitly named in 41 studies. Table 8 shows an overview of the identified companies and the number of studies reporting them. The most frequently identified company was Ericsson (cf. [S1], [S3], [S26]), followed by the Norwegian Public Service Pension Fund (cf. [S28], [S31], [S43]). Other companies repeatedly studied were F-Secure (cf. [S8], [S68]) and Nokia (cf. [S2], [S79]). The remaining 17 companies were mentioned only in one study. As a result, we notice that research on large-scale agile development is mainly empirical and practice-oriented, as almost 80% of the studies examine companies in their respective analyses. We note that a significant portion of the research was conducted with Ericsson, accounting for nearly 17% of all selected studies. We notice that many studies provide superficial case descriptions, making it difficult to generalize and compare results.

Table 8: Overview of the identified case companies

Company	No. of studies
Anonymous	67
Ericsson	23
Norwegian Public Service Pension Fund	8
F-Secure	2
Nokia	2
ABB	1
Apontador	1
BBC	1
Caelum	1
Cisco	1
Comptel	1
Dell	1
Information Mosaic	1
Intel	1
Kentico	1
Paf.com	1
Rovsing	1
SAP	1
SimCorp	1
Spotify	1
ThoughtWorks	1
Universo Online	1

Figure 4 shows the countries where the headquarters of reported companies are located, with circles indicating the number of companies in a given country. We identified the geographic location of 58 companies. Although most companies are located in Europe, large-scale agile development is a relevant practical topic on all continents. Most companies come from Germany, followed by the United States, Norway, and Sweden. Like the State of Agile survey [31], we note a concentration of industry relevance of the topic in Europe and North America. Based on our selection criteria (see Section 3.2.2), we identified fewer companies from North America compared to the State of Agile survey [31] as we excluded many experience reports and opinion articles from North America.

[4]There is some likelihood that some unnamed companies are duplicates. Many case descriptions were too superficial to allow clear identification.

Figure 4: Geographical distribution of the case companies

Figure 5 shows the distribution of the companies across their sectors. While 146 companies come from 10 different sectors, we could not identify the respective sectors for 12 companies. Almost a third of all reported companies come from the information technology sector, followed by the financial and public sectors. Our findings align with the State of Agile survey [31] stating that a large proportion of the companies using (large-scale) agile methods come from the information technology and financial sectors. Notably, 44% of the companies surveyed come from the information technology and finance sectors, while 46.84% of our companies are from these sectors.

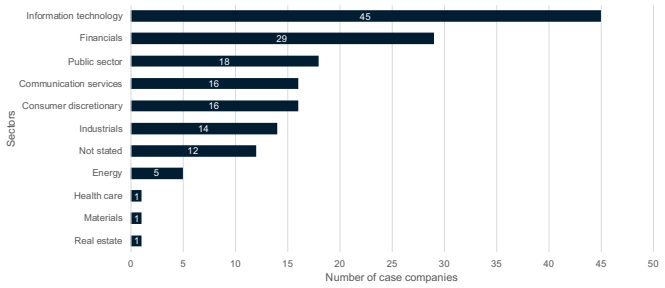


Figure 5: Distribution of the case companies according to their sectors

Figure 6 shows the distribution of the reported companies over their sizes. While we were able to determine the organizational size of 89 companies, we could not reveal the size of 69 companies. Most companies employ $>20,000$ employees, followed by companies with $\leqslant 1,000$ employees. Comparing our results to those of the State of Agile survey [31], we can see some similarities and differences. Like the survey [31], the majority of reported companies are small ($\leqslant 1,000$ employees) or very large ($>20,000$ employees). While these two groups account for 66% of survey respondents, they account for 65.17% of the reported companies in this study. The majority of the companies in our sample are very large, followed by small companies, which is precisely the opposite of the survey [31]. One reason for this could be that the survey [31] covers both the small- and large-scale adoption of agile methods. Our research exclusively considers the large-scale application of agile practices, which may be more relevant for large companies than for small companies with fewer scaling issues.

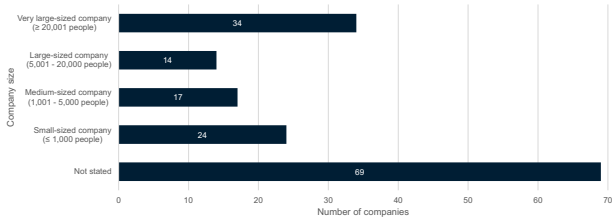


Figure 6: Distribution of the case companies according to their sizes

4.1.2. *Understanding of large-scale agile development*

Since there are several definitions of large-scale agile development in the literature (see Section 2.2.1) and there is no consensus on the actual meaning of the term [35, 90], we were curious what is meant by "*large-scale agile development*" in the literature. To this end, we used the classification by Fuchs and Hess [42], i.e., the adoption of agile methods in large multi-team settings ("*large agile multi-team settings*") with at least two agile teams working on a single product or the usage of agile practices in organizations as a whole ("*organizational agility*"), to analyze the large-scale agile efforts of the reported companies. Figure 7 shows the categories that have been reported. Nearly half of all companies applied agile practices in organizations as a whole, while 39.24% of all companies used agile methods in large multi-team settings. An example

for organizational agility is provided by Fuchs and Hess [S21] who conceptualize the agile transformation process through the lens of socio-technical systems theory by analyzing the large-scale agile transformations of two companies from the financial and consumer discretionary sectors. An example for large agile multi-team settings is shown by Šāblis and Šmite [S16] who study inter-team coordination mechanisms of a large-scale agile development program of the Norwegian Public Service Pension Fund with eight teams. In four companies, both categories were reported. For instance, Power [S7] explains the notion of "*agile at scale*" by showing an example of Cisco using agile practices in the whole organization and having multiple development efforts with several agile teams working together. For 16 companies (cf. [S107], [S117]), we did not determine the category due to superficial context information.

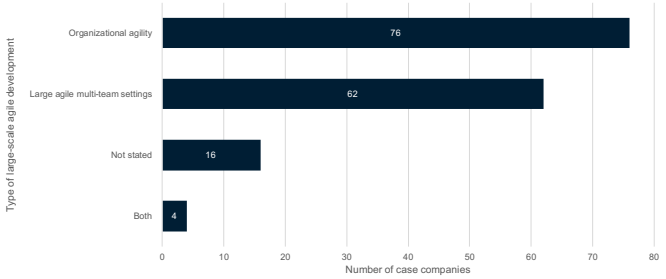


Figure 7: Type of reported large-scale agile development

In the following, we delve into the selected studies and provide more in-depth information in light of both categories of large-scale agile development reported in the companies.

4.1.3. Organizational agility

To understand the trend of organizational agility, we analyzed when the case companies started their large-scale agile transformations. Figure 8 shows the distribution of when case companies started their large-scale agile transformations. We derived information on the agile transformations of 20 case companies. The first two observed transformations towards organizational agility began in 2006 at F-Secure (cf. [S68]) and Paf.com (earlier Eget) (cf. [S88]), when the company-wide adoptions of Scrum were initiated. While nine transformations started between 2006 and 2012, 11 transformations began between 2014 and 2017, indicating a growing number of transformations in recent years. Comparing these numbers with the number of studies published (see Section 4.2.1), we perceive the congruent interest from industry and academia in the topic.

4.1.4. Large agile multi-team settings

We identified 110 agile development efforts[5] with multiple teams. There were also some companies with more than one effort. For instance, we uncovered 11 efforts with 6–40 agile teams at Ericsson, reported in 20 studies (cf. [S1], [S3], [S26]).

[5]There is some likelihood that some large-scale agile development efforts are duplicates, e.g., same settings at different times. Some case descriptions were too superficial to allow unambiguous identification.

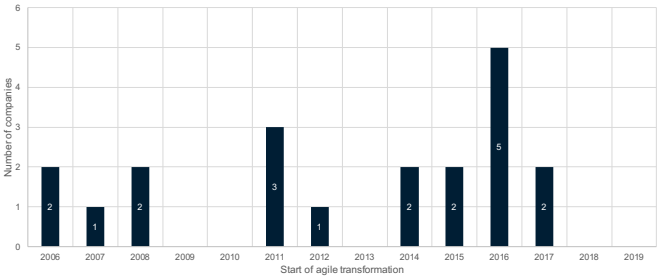


Figure 8: Start of the agile transformations of the case companies

Another example is provided by Bick et al. [S51], who describe the inter-team coordination of five large agile multi-team settings at SAP, involving 4–13 teams.

Based on our ambition to gain a deeper understanding of what the authors of the selected studies mean by the "*scaling of agile methods*", we examined the case descriptions in terms of possible scaling and complexity factors of large agile multi-team settings. Figure 9 provides an overview of the identified factors. We identified six scaling and complexity factors associated with large agile multi-team settings, i.e., organizational size, organizational distribution, product size, number of collaborating companies, number of customers, and budget. Organizational size, i.e., the number of agile teams working together or the number of people in the development effort, was the most frequently observed factor. For instance, Paasivaara and Lassenius [S79] present a case study of scaling Scrum in a large globally distributed software project at Nokia. The study expresses the scale and complexity of the case project by describing that it scaled from two collocated Scrum teams to 20 teams, i.e., the number of agile teams working together, and now employs 170 people, i.e., the number of involved persons in the development effort. Organizational distribution of large agile multi-team settings was the second most frequently identified factor. Here, the studies specify the number of sites a development effort has and provide information regarding the geographic distribution of the development effort. For example, Usman et al. [S37] examine how a large-scale distributed agile project at Ericsson performs effort estimation. The study further delineates the project's organizational distribution by stating that the project is distributed across Sweden, India, Italy, the United States, Poland, and Turkey. The third most frequently cited factor is related to the complexity of the developed system, i.e., product size (measured in lines of code), number of subsystems, and number of requirements and features. For instance, Moe et al. [S67] investigate the intra- and inter-team knowledge sharing in a large-scale distributed agile project at Ericsson, developing a product with 10–12 million lines of code and 30 subsystems. In the case of 14 large agile multi-team settings, the studies did not provide information on scaling and complexity factors. For instance, a multiple case study of five software-intensive companies by Olsson and Bosch [S98] reveals challenges associated with collecting and using customer feedback. Although we can infer from the context of the study that it analyzes large agile multi-team settings of five companies, we did not find fur-

ther descriptions of their characteristics. Moreover, the number of companies working together was an additional factor. For example, Uludağ et al. [S104] investigate the collaboration between architects and agile teams of a large-scale agile development program at a German consumer electronics company. The reported large-scale agile program includes 62 persons employed in four companies, i.e., consumer electronics and three suppliers providing external support for their third-party systems. In two large agile multi-team settings, the number of customers/users of the developed software product was cited as a factor. For example, a large-scale agile program at Ericsson creates a complex system used by over 300 operators around the world (cf. [S3], [S38], [S50]). For instance, a large-scale agile endeavor implements a new office automation system for the Norwegian Public Service Pension Fund serving 950,000 customers with various types of services (cf. [S43], [S52]). Budget was reported in one setting, namely at the Norwegian Public Service Pension Fund, which is one of the most extensive IT programs in Norway with a budget of about 140 million EUR (cf. [S28], [S31], [S43]). Comparing our findings with the scaling and complexity factors of Ambler [7], we note some similarities and differences. Like Ambler [7], we believe that organizational size is the primary scaling and complexity factor for characterizing the term "*scaling of agile methods*". Like Ambler [7], we believe that other important factors are organizational distribution and system complexity. Unlike Ambler [7], we identified the number of collaborating companies, the number of customers, and the available budget for development efforts as additional scaling and complexity factors.

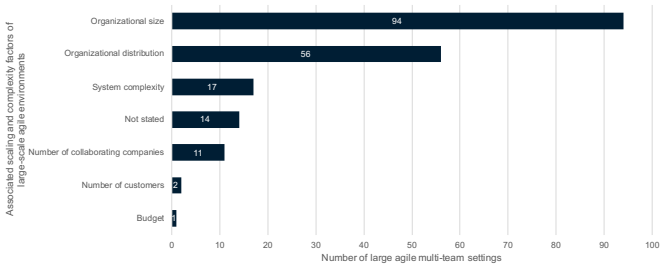


Figure 9: Scaling and complexity factors of large-scale agile environments

Since organizational size was the main reported factor, we were curious to analyze the sizes of the agile multi-team settings. Figure 10 shows the distribution of the number of people and teams involved in the agile multi-team settings. Of the 110 reported agile multi-team settings, we determined the number of people involved for 70 settings and the number of teams for 62 settings. We could not determine the number of people engaged for 40 settings and the number of teams for 48 settings. Most of the reported agile multi-team settings were large and included 11–100 people or 2–9 teams, followed by very large settings that involved 101–8,000 people or 10–40 teams. Based on our understanding of large-scale agile development (see Section 2.2.1) and excluding projects with <2 teams, we found no small-scale projects. Our results show some notable differences from the State of Agile survey [31]. While 70% of the settings (excluding settings without numbers) had $\leqslant$100 employees, 33% of all survey respondents reported that their software organizations had $<$100 employees. While 36% of the survey respondents indicated software organization sizes with 101–1,000 employees, this was the case for 46.67% of agile multi-team settings in this study. A significant difference between this study and the survey is observed for software organization sizes with $>1,001$ employees. While 31% of the survey respondents reported software organization sizes of $>1,001$ people, we identified only one very large agile multi-team setting at Cisco with 8,000 employees (cf. [S7]). This discrepancy may be due to our selection criteria related to experience reports and opinion papers (see Section 3.2.2) led to the exclusion of companies in North America with likely larger sizes.

Finally, we were interested in exploring the applied development and scaling approaches reported in the large agile multi-team settings, visualized in Figure 11. SoS is the most commonly used scaling approach reported, followed by SAFe and LeSS. The Spotify Model was described in three settings, followed by DAD in one setting. Comparing our numbers with those from the State of Agile survey, we see some similarities and differences. While the survey showed SAFe as the most commonly used scaling approach among respondents (35% of their respondents), only 8.18% of the large agile multi-team settings reported using SAFe. For SoS, the numbers are similar as 18.18% of all settings used this scaling approach, while 16% of the survey participants applied it. In addition, 4% of the survey participants reported using LeSS, while 7.27% of the settings in this study used LeSS. While we identified the adoption of DAD in only one setting, 4% of all survey participants adopted it. Scrum was the most frequently cited development approach, followed by Kanban and XP. Re-comparing our results with the survey data confirms that Scrum is by far the most frequently used development approach, i.e., 58% of the survey respondents mentioned Scrum. In contrast, Scrum was used in 72.7% of all large agile multi-team settings reported in this study. While 10.91% of the settings reported using Kanban, that number accounted for 7% of all survey respondents. About half of the settings stated the combined usage of development and scaling approaches. For instance, a large multi-team project at F-Secure, which included $>$10 teams and $>$140 stakeholders, used a combination of Scrum, SoS, and an early version of SAFe (cf. [S8], [S68]). Uludağ et al. [S87], for example, describe a unit with three agile teams who worked with other groups to develop an integrated sales platform for multiple sale distribution channels, using LeSS extended with some XP practices.

4.2. Publication trends and characteristics of existing research on large-scale agile development

To map publication trends and characteristics of extant research on large-scale agile development, we chose a set of variables focusing on each study's publication and bibliographic data. Below, we detail the key facts drawn from our analysis.

4.2.1. Distribution of studies over time

Figure 12 presents the distribution and cumulative sum of selected studies, providing a clear message on an apparent increasing trend in publications on large-scale agile development.

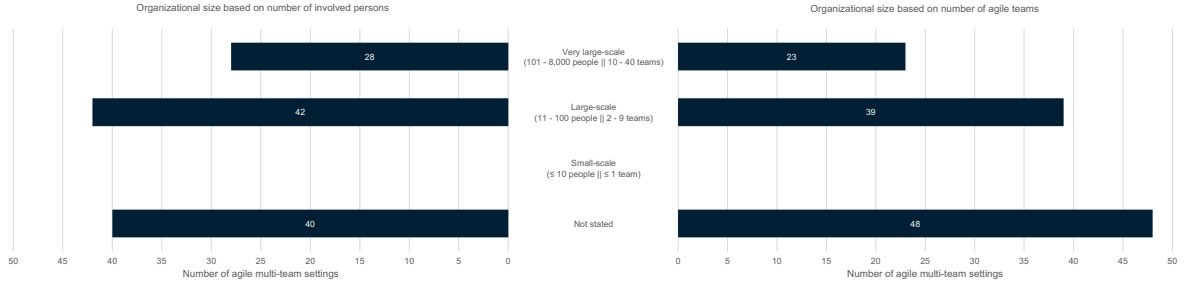


Figure 10: Distribution of organizational sizes of the large agile multi-team settings in terms of number of involved persons and agile teams

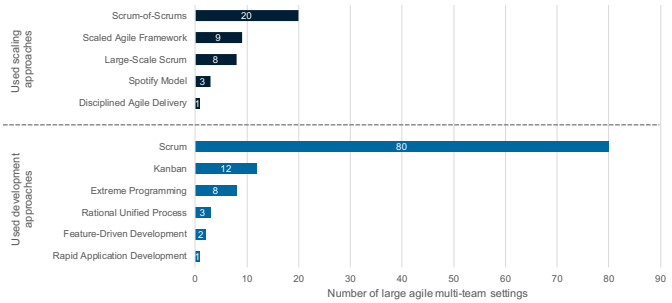


Figure 11: Applied development and scaling approaches in the large agile multi-team settings

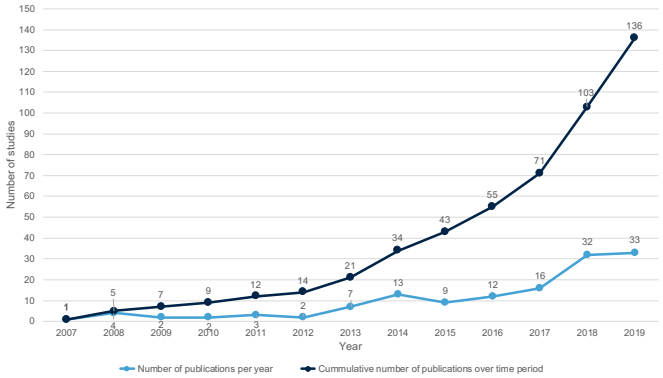


Figure 12: Publications on large-scale agile development from 2007 to 2019

In general, the number of selected studies on this topic has increased from 2007 to 2019, with slight fluctuation. This trend indicates that the large-scale application of agile methods is receiving increasing attention from the research community. Except for 2007, at least two studies were published during the observation period. From 2013 onwards, a growing trend can be observed with at least seven studies published per year, which is a giant leap compared to the years before 2013. Two reasons for this could be that the agile software development community initiated an increasing paradigm shift towards the adoption of agile methods in large projects in 2013 [33, 41] and that the International Workshop on Large-Scale Agile Development was held for the first time in 2013. After 2013, 84.55% of the selected studies were published in the last six years, indicating that the topic of large-scale agile development is relatively infancy when compared to the history of the software engineering discipline and is becoming increasingly attractive from a scientific perspective. The growing trend towards the topic culminated in 2018 and 2019 when the number of publications doubled compared to previous years, accounting for a total of 47.79% of the selected studies. One possible explanation for this is that additional workshops contributing to large-scale agile development research were initiated, e.g., the International Workshop on Autonomous Agile Teams in 2018 and the International Workshop on Agile Transformation in 2019. Based on these observations, we expect this trend to continue.

4.2.2. Most active countries in large-scale agile development research

Figure 13 shows the countries that are most active in large-scale agile development research. We identified 22 states contributing to large-scale agile development research. The majority of the articles are from Europe, accounting for 88.97% of all publications. The research theme of large-scale agile development received considerable interest in Scandinavia, with 74 papers. Accordingly, the most actives countries in this research area are Finland, Norway, and Sweden. The fourth and fifth most active countries are Germany and the United Kingdom. The remaining 17 states have fewer than 10 publications and contributed only 32 articles. From the available data, we conclude that large-scale agile development is a globally relevant research topic. Although Dingsøyr et al. [34] revealed that the United States and Canada contributed to research on agile software development with 448 out of 1,551 selected studies, we identified only three studies on large-scale agile development. A plausible explanation for this phenomenon is that we identified numerous experience reports and opinion papers from both countries that were excluded by our selection criteria (see Section 3.2.2) (cf. [50], [60], [85]). Comparing the map of most active countries in large-scale agile development research (see Figure 13) with the geographic location of the reported companies (see Figure 4), a considerable overlap can be observed. A logical reason for this observation is that researchers tend to work with companies that are located in closer proximity.

4.2.3. Publication channels

To examine the pertinent publication channels for large-scale agile development research, we collected data on publication types, venues, and domains for each selected study. Figure 14 shows the proportion of publication types for the 136 selected

Figure 13: Map of most active countries in large-scale agile development research

studies. The predominant publication type is that of conference papers, being almost as much as the combination of journal articles and workshop papers. Such a high number of journal and conference papers may indicate that large-scale agile development is becoming a more mature research area despite its relatively young age. The relatively small number of workshop papers suggests that researchers prefer more scientifically rewarding publication types, e.g., journals and conferences.

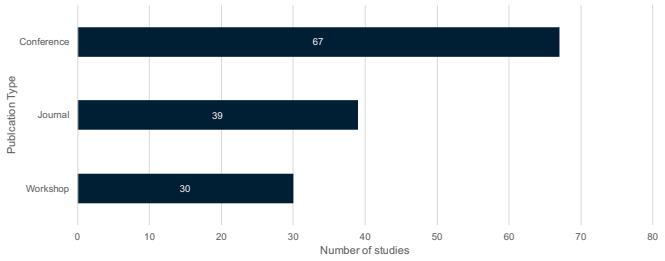


Figure 14: Distribution of publication types

Figure 15 shows an annual distribution of the publication type facet. After 2013, an increasing trend can be observed for all publication types, culminating in 2018 and 2019. This observation may further confirm that large-scale agile development is turning into a more mature field, with more fundamental and comprehensive studies published in recent years. Although journals were the predominant publication type between 2007 and 2011, we can observe that conferences and workshops became increasingly prevalent from 2012 onwards, as more venues on the topic were initiated or existing venues included large-scale agile development in their research program.

Figure 16 presents the distribution of selected studies across publication domains. We identified eight publication domains that cover research related to large-scale agile development.

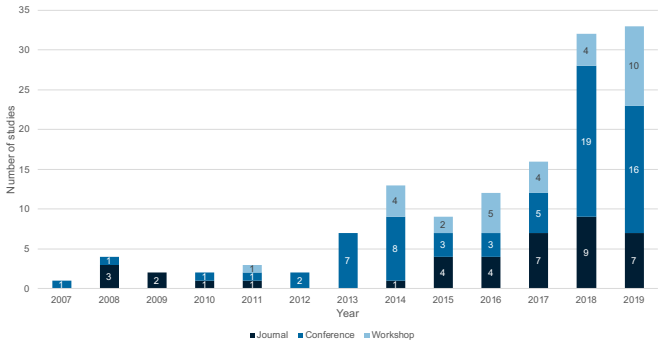


Figure 15: Distribution of publication types over time

Unsurprisingly, most of the publications are covered by publication venues from the software engineering research domain, followed by the information system and project management research areas. Nine articles were published in venues covering various research topics, e.g., IEEE Access embracing all IEEE fields of interest or Procedia Computer Science containing conference proceedings on all computer science topics. Although most of the studies stem from the software engineering field, we recognize a growing interest in large-scale agile development research from further research communities, e.g., information systems, project management, and enterprise computing.

To corroborate this observation, we show in Figure 17 how the number of publications has evolved across the identified publication domains. Large-scale agile development is primarily and expectedly domicile in the software engineering field, with at least one published study per year and a peak in 2019 with 22 publications. Research on large-scale agile development has also aroused an early interest in publication venues related to information systems. After 2013, we observe a grow-

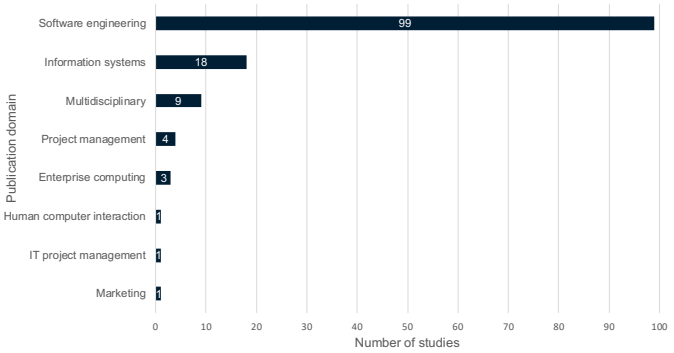


Figure 16: Distribution of publication domains

ing interest in the topic outside of software engineering. While only two articles were published in venues unrelated to software engineering between 2007 and 2012, further 35 articles were published between 2014 and 2019, representing 94.59% of all studies outside of software engineering. These numbers are coherent with our observations. We notice a general interest in the topic in various academic fields in recent years, as the large-scale adoption of agile methods has far-reaching implications for companies, which in turn is very interesting for many scientists to investigate from an empirical point of view.

A look at the specific targeted publications venues shows that research on large-scale agile development is published in various venues. The 136 selected studies are distributed across 46 publication venues (see Appendix A), including 17 journals, 23 conferences, and six workshops, indicating that research on large-scale agile development is receiving wide attention in the research community. Table 9 presents the top-10 venues with the highest number of publications. While eight of the top-10 publication venues belong to the software engineering research field, two conferences are from the information systems research field. The top six venues account for 50% of all publications, while the remaining 40 venues cover half of all studies. The top venues are the International Conference on Agile Software Development, followed by the International Workshop on Large-Scale Agile Development. Both venues have at least twice as many publications as the third-largest contributor, the International Conference on Global Software Engineering. The top contributing journals are IEEE Software and Information and Software Technology. The publication venues' distribution indicates that contributions are published primarily at the International Conference on Agile Software Development and International Workshop on Large-Scale Agile Development, which is unsurprising as they deal exclusively with this topic. Researchers and practitioners can consider these venues as an entry point to explore the state of the art in large-scale agile development research and gain easier access to the community. We note that several studies are present in leading software engineering journals, e.g., IEEE Software, Information and Software Technology, and Empirical Software Engineering. Since six publication venues cover half of all publications, we assume that researchers primarily submit their research to a few specific venues. The publication venues' distribution shows a long tail of 27 venues, each with one published study.

4.2.4. Research types and methods

Figure 18 shows the distribution of selected studies against their research types. We only identified three research types of the six research types specified by Wieringa et al. [103] (see Table 5), namely evaluation research, solution proposal, and philosophical papers. We note that 110 articles report on evaluating solutions applied in practice, typically through case studies, mixed methods, surveys, or similar empirical methods. Solution proposals were the next most frequently covered research type presenting either novel solutions or significant extensions of existing solutions. Philosophical papers that offer new perspectives on existing things by framing the large-scale agile development field using conceptual frameworks or taxonomies were the third most commonly identified research type. Based on our selection criteria (see Section 3.2.2), we could not identify any experience and opinion papers since they were excluded during our selection process. We could also not identify any validation research studies in which researchers investigate novel techniques that have not yet been implemented in practice. This observation would also have been atypical as most of the selected articles are concerned with identifying, describing, and evaluating the practical application of agile methods on a larger scale. The fact that evaluation research is widely used positively impacts the potential for transferring current research findings to the industry. The prevalence of evaluation research studies is natural in a phenomenon such as large-scale agile development, which is driven primarily by industry instead of resulting from the context of a research lab.

Figure 19 shows the evolution of research types over time. We notice that evaluation research studies dominate large-scale agile development research throughout the period considered, especially before 2013, when 12 out of 14 papers belong to this category. Accordingly, before 2013, only two articles were published from the solution proposal research type. After 2013, an increasing tendency towards all three identified research types can be observed. While between 2014 and 2019, at least six evaluation research papers were published each year, at least three articles per year were published combining philosophical papers and solution proposals. As a result, we observe a promising development in large-scale agile development research, i.e., researchers evaluate existing techniques in practice and develop new solutions based on their observations and frame the observed phenomena using conceptual models.

To investigate research methods applied in large-scale agile development, we visualize the research approaches used in the selected studies in Figure 20. Most studies are empirical and include case studies, mixed methods, ground theory, survey, design and creation, and action research. By far, the most prevalent research method in large-scale agile development is case study research, followed by (systematic) literature review/(systematic) mapping study, mixed methods, and grounded theory. We could not identify the applied research method in seven articles, as it was neither described nor derivable. Only three papers are theoretical, mainly in the form of conceptual models. For instance, Carroll and Conboy [S113]

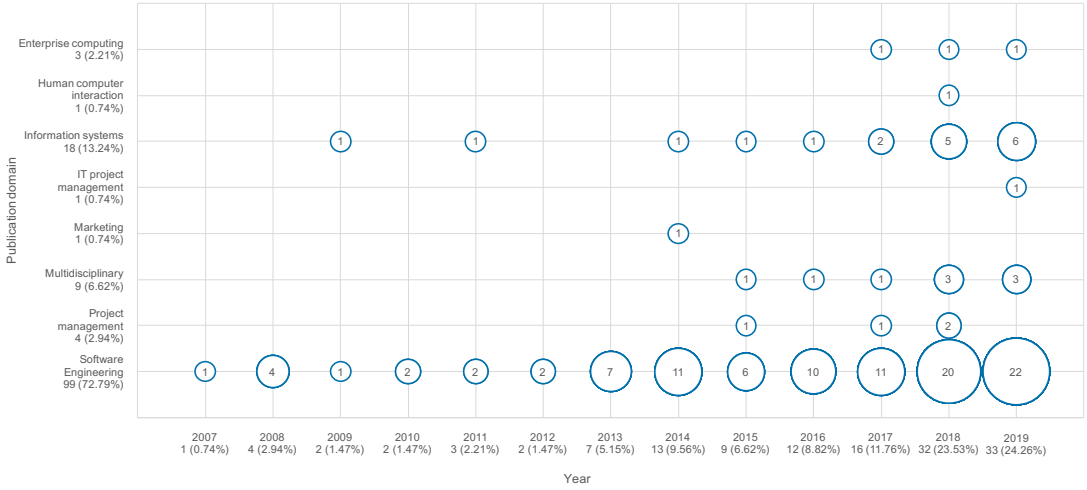


Figure 17: Number of studies per publication domain over time

Table 9: Top-10 publication venues ranked by number of submitted studies

#	Publication source	Type	Domain	No.	%
1	International Conference on Agile Software Development	Conference	Software engineering	21	15.44
2	International Workshop on Large-Scale Agile Development	Workshop	Software engineering	20	14.71
3	International Conference on Global Software Engineering	Conference	Software engineering	9	6.62
4	IEEE Software	Journal	Software engineering	6	4.41
5	Information and Software Technology	Journal	Software engineering	6	4.41
6	International Workshop on Autonomous Teams	Workshop	Software engineering	6	4.41
7	Empirical Software Engineering	Journal	Software engineering	5	3.68
8	Hawaii International Conference on System Sciences	Conference	Information systems	5	3.68
9	Americas Conference on Information Systems	Conference	Information systems	4	2.94
10	Journal of Systems and Software	Journal	Software engineering	4	2.94

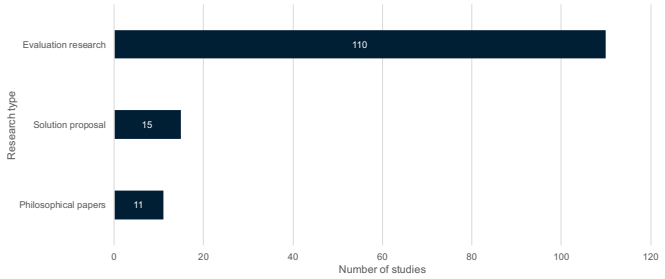


Figure 18: Distribution of research types

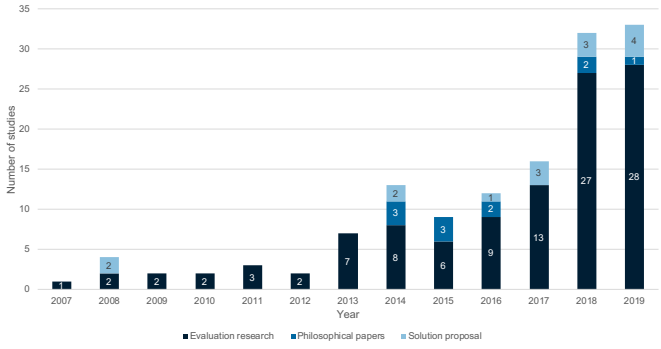


Figure 19: Number of studies per research type over time

apply normalization process theory to theorize the process of large-scale agile transformations by identifying existing assumptions about these transformations and explaining what factors enable their normalization. Sweetman and Conboy [S127] use the complex adaptive systems lens to critically appraise current thinking on portfolio management in an agile context. We identified only one action research paper, namely the from Pries-Heje and Krohn [S95], that describes the adoption of SAFe at SimCorp and the resulting implementation challenges using a participatory action research project. The results show that the body of knowledge on large-scale agile development is mainly empirical and is still at an exploratory stage. A high percentage of articles are case studies that are sparsely based on grounded theory or theory-building methods.

To reveal trends, we visualize the annual distribution of applied research approaches over time in Figure 21. Figure 21 confirms the prevalence of case study research from a temporal perspective, i.e., case study research was applied throughout the entire observation period and shows a nearly constant increase over time, peaking in 2018 and 2019. Between 2007 and 2013, 76.19% of the studies were based on case study research. At the same time, only four out of the eight identified research approaches, i.e., case study, design and creation, grounded theory, and survey. As of 2014, the remaining four research meth-

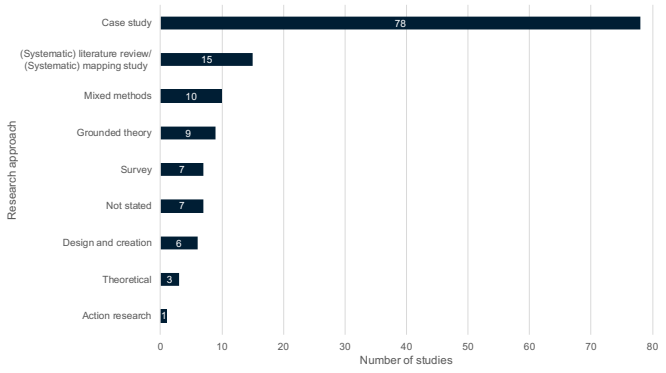


Figure 20: Distribution of applied research approaches

ods, i.e., action research, mixed methods, (systematic) literature review/(systematic) mapping study, and theoretical, were applied. The most diverse usage of research approaches was observed in 2019, when seven different research methods were used. The increasing use of various research methods validates our view that large-scale agile development is a multifaceted research field necessitating the investigation of the observation object from different scientific angles. This trend demonstrates the increasing demand from companies for academic support, which fuels the growing research interest of scholars.

4.2.5. Research outcomes

Figure 22 shows the distribution of research contributions of the 136 selected studies. A total of 115 papers contribute in the form of lessons learned and guidelines. They are rather observational by analyzing and describing the application of agile practices in large-scale projects. Conversely, the remaining studies contribute models, frameworks/methods, and theories. For instance, Qumer and Henderson-Sellers [S6] propose a new framework called Agile Software Solution Framework to support the adoption and improvement of agile methods in large-scale software projects. Using the human systems dynamics model, Power [S7] presents a new model to determine when scaling agile practices is appropriate for large organizations. Bass [S18] provides another example and uses the Glaserian grounded theory approach to study and explain artifact inventories used in large-scale agile offshore software programs. None of the selected studies contribute to developing new or improved tools. This observation suggests that further research is needed that develop conceptual models and theories to strengthen the theoretical foundations of large-scale agile development research and create rigorously developed frameworks, methods, and tools to assist practitioners.

Figure 23 shows an annual distribution of the contribution type facet. Most articles focused on deriving lessons learned from observations made on large-scale agile endeavors between 2007 and 2019. During this period, at least one lessons learned study was published annually, peaking in 2018 and 2019. Starting in 2014, with at least two published studies per year, a clear tendency towards creating frameworks/methods, models, and theories can be observed. This observation indicates growing empirical evidence owing to the rising interest in the research community and the growing body of knowledge.

To spot possible patterns between the publications domains' preference for made contribution types and applied research approaches, Figure 24 displays the relationship between these three data items. We can see that in almost all domains, except for IT project management, deriving lessons learned from observations made is the most common contribution type. We note that most diverse contributions have been made in the software engineering field, confirming its multidisciplinary nature (cf. [45]) even in the context of large-scale agile development. Concerning the relationship between publication domains and research methods, we notice that case study research is preferred in most domains. Although case study research is the most commonly used approach in the information systems and software engineering domains, we realize that these two domains use a wide variety of research methods.

To better understand the relation between the research outcome and the applied research type and approach of each study, we visualize the relationship between these three data items in Figure 25. The predominance of the selected articles applies case study research to evaluate the usage of agile methods in large-scale projects and derive a set of lessons learned based on the analysis of the research findings. In terms of evaluation research, the remaining articles apply various research methods, mainly (systematic) literature reviews/(systematic) mapping studies, mixed methods, and surveys, to draw some lessons learned. Further 17 papers use case study research, design and creation research, grounded theory, and mixed methods research to make contributions in the form of taxonomies, conceptual frameworks, and theories. For instance, by applying grounded theory, Santos et al. [S44] create a conceptual model that aims to explain the dependence of adequate knowledge sharing across agile teams on the purposeful application of agile practices and organizational conditions and stimuli. Turetken et al. [S19] propose a maturity model to assess the degree of adopting SAFe in companies using design science research. Rolland et al. [S73] employ a mixed methods research design consisting of a literature review and case study to create a conceptual model for examining the underlying assumptions of large-scale agile development. By comparing contribution types and research types and methods, the under-representation of the usage of systematic literature reviews/systematic mapping studies and surveys becomes apparent. Only 16.18% of the studies use (systematic) literature reviews/(systematic) mapping studies and surveys to contribute in terms of lessons learned. These approaches are non-existent in other contribution types. This observation indicates a research gap in creating new conceptual models or adapting existing theoretical models from different research domains to explain the various aspects of large-scale agile development based on quantitative surveys and statistical analyses.

4.2.6. Research data types

While 121 of the selected papers are primary studies and another 15 articles are secondary studies, we did not find any tertiary studies. Six secondary studies provide ad hoc (systematic) literature reviews on challenges and success factors

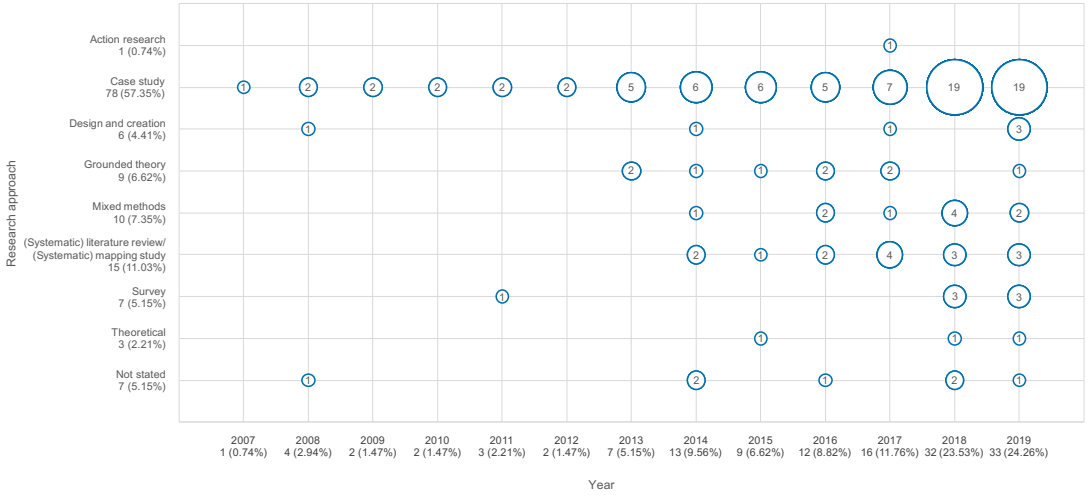


Figure 21: Number of studies per research approach over time

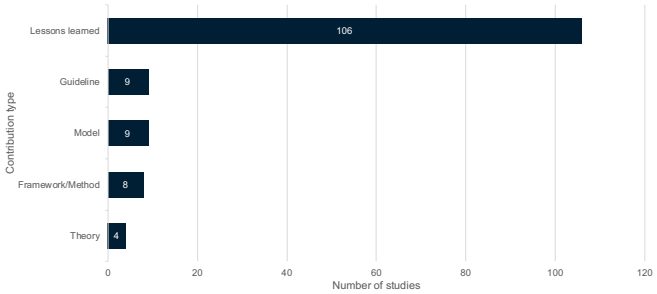


Figure 22: Distribution of research outcomes

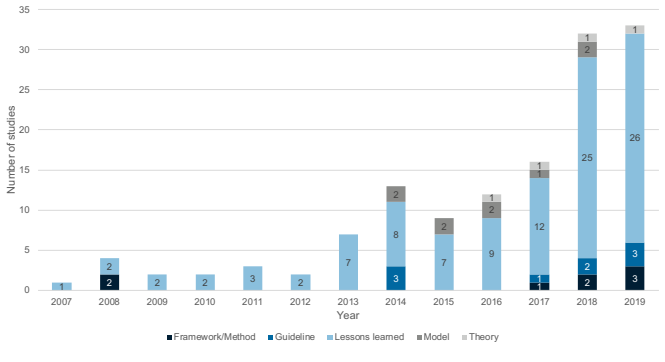


Figure 23: Distribution of research outcomes over time

of applying agile methods in large (and distributed) software projects (cf. [S10], [S22], [S47], [S89], [S135], [S136]). Four secondary studies compare and analyze various scaling agile frameworks based on (systematic) literature reviews (cf. [S9], [S17], [S57], [S108]). Based on a literature review, one secondary study identifies roles responsible for inter-team coordination of agile teams (cf. [S20]). One secondary study reveals a set of motivators for large-scale adoption of agile methods from a management perspective (cf. [S65]). One secondary study provides a multi-vocal literature review describing the benefits and challenges of adopting SAFe (cf. [S132]). One secondary study conceptualizes coordination in large-scale agile development from a multi-team systems perspective (cf. [S30]). One secondary study provides a set of challenges that harm quality requirements in large-scale distributed agile projects and proposes a set of solutions to overcome them (cf. [S134]).

Figure 26 shows an annual distribution of the research data type facet. Alongside the sharp increase in the number of published studies over the past decade, secondary studies have also increased in recent years. Specifically, all secondary studies were published after 2013. The increase of secondary studies indicates the growing body of knowledge and maturity of the large-scale agile development research field.

4.3. Seminal studies

As suggested by Dingsøyr et al. [34] and Nerur et al. [71], we identified seminal works as they facilitate understanding and exploration of the intellectual structure of the large-scale agile development research field. Like Herbold et al. [48], we extracted data on citation numbers from Google Scholar to define our criterion for seminal publications and consider the top 10% of the studies with the most citations as influential (see Table 10). The 13 seminal publications:

1. discuss effects, issues, benefits, and success factors related to the large-scale introduction of agile practices in plan-driven organizations (cf. [S1], [S2], [S5], [S22], [S92]),
2. describe experiences in applying agile practices based on Scrum to large, globally distributed software development programs (cf. [S34], [S103]),
3. present the role of communities of practices and the associated challenges and success factors as part of large-scale agile transformations (cf. [S3]),
4. propose a scaling framework for creating new agile software development processes to develop large and complex software applications (cf. [S6]),
5. propose a framework for guiding software process improvement activities concerning agility in large software product development organizations (cf. [S24]),

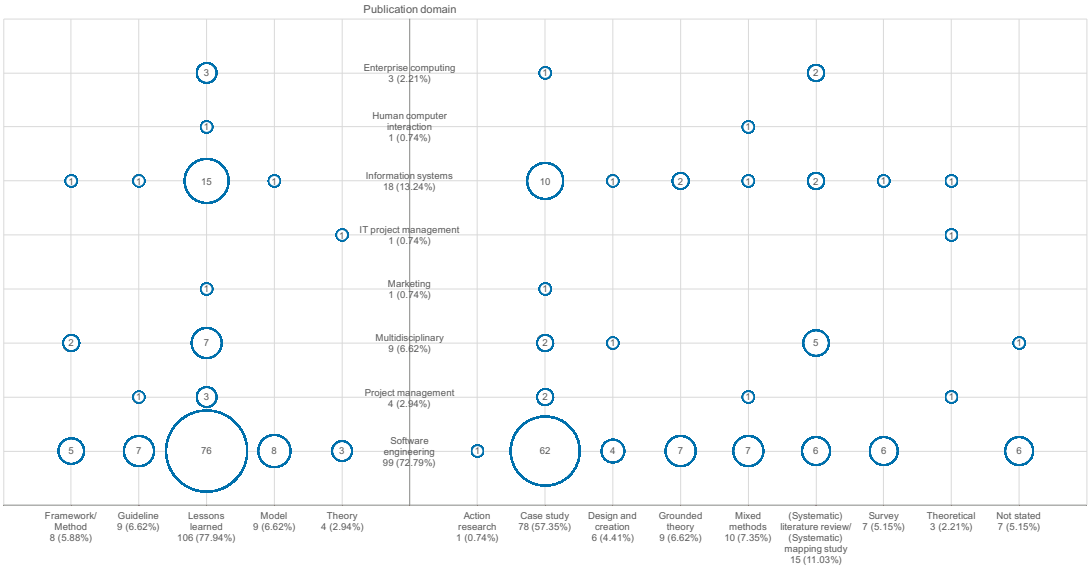


Figure 24: Mapping of publication domains against contribution types and research approaches

6. elucidate the adaption of agile methods in terms of customer involvement, software architecture, and inter-team coordination in a large-scale agile program (cf. [S43]),
7. report the application of the scaling approach SoS and its associated challenges and success factors in large-scale distributed Scrum projects (cf. [S53]), and
8. describe the adoption of portfolio management practices and the associated benefits and side-effects in organizations applying agile methods at large-scale (cf. [S112]).

4.4. Research streams and research agenda in large-scale agile development

As a common goal of systematic mapping studies is to assess the state of the art and maturity level of a research area [58, 82], we present the general structure, central research themes, and research gaps of the large-scale agile development research field. Below we present our key findings related to the identified research streams and gaps before discussing each of them.

4.4.1. General overview

We explored research clusters to identify pertinent research themes and structured the selected studies according to these themes. Figure 27 provides an overview of the identified research streams and shows the number of studies assigned to each stream. We identified 10 research streams, namely *Agile architecture*, *Agile planning*, *Agile portfolio management*, *Agile practices at scale*, *Communication and coordination*, *Global and distributed software engineering*, *Large-scale agile transformations*, *Scaling agile frameworks*, *Taxonomy*, and *Team autonomy*. While some research streams have enjoyed very little research interest, e.g., *Taxonomy* or *Agile portfolio management*, researchers have devoted a great deal of attention to investigating the scaling of agile practices, i.e., *Agile practices at scale*, studying the coordination of large agile multi-team settings, i.e., *Communication and coordination*, and analyzing the adoption of scaling frameworks, i.e., *Scaling agile frameworks*.

Table 11 shows a tabular representation of the research streams, including their sub-topics. While most research streams exhibit numerous endeavors and clusters dealing with specialized sub-topics, some research streams scarcely show any specific sub-research cluster. For instance, in the research stream *Communication and coordination*, we identified several sub-topics, e.g., *Communication mechanisms, Team performance*, and *Knowledge networks*, while we did not observe any specialized sub-themes in the *Team autonomy* and *Taxonomy* research streams. In almost all research streams (except *Taxonomy*), we observe several studies that derived several lessons learned in the form of challenges and success factors based on their investigations. The complexity and the number of sub-clusters identified per research stream are logically reflected in the number of studies assigned to a stream. While only three papers were assigned to the *Taxonomy* research stream with no specialized sub-topic, nine sub-streams were identified in the *Agile practices at scale* research stream.

Figure 28 visualizes how the number of studies assigned to the research streams has developed over time. The earliest researches on large-scale agile development were conducted between 2007 and 2008 and are related to the *Agile practices at scale*, *Global and distributed software engineering*, and *Scaling agile frameworks* research streams. Tessem and Maurer [S126] represent the first reported paper on large-scale agile development belonging to *Agile practices at scale* research stream and describing the adaptation of Scrum for a project involving 70 people. The initial contributions to the four research streams: *Agile architecture*, *Team autonomy*, *Communication and coordination*, and *Taxonomy* were published between 2013 and 2016, when a great interest in the large-scale agile development research topic emerged. After 2013, there has been a general

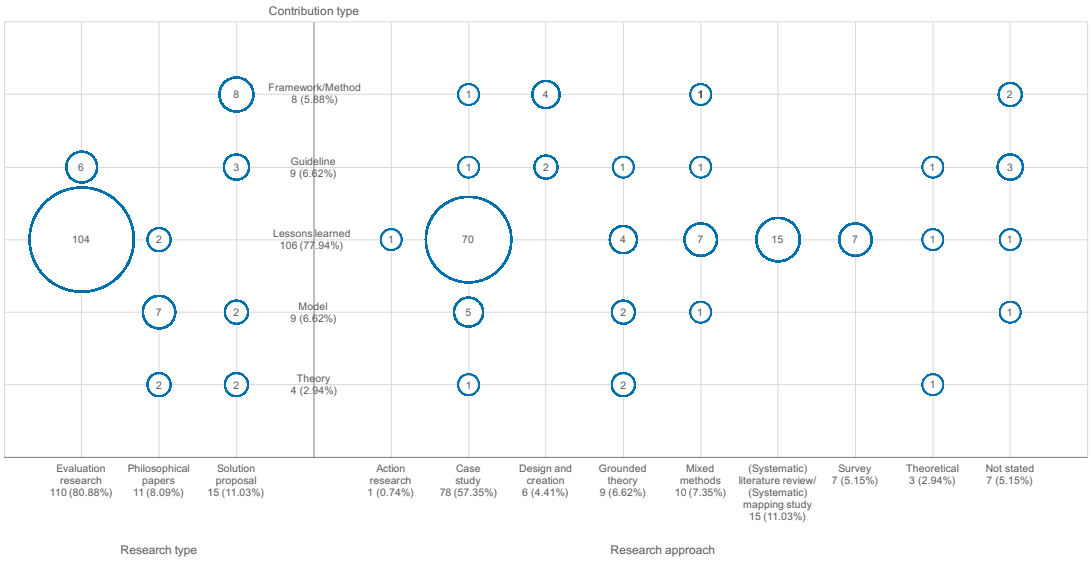


Figure 25: Mapping of contribution types against research types and approaches

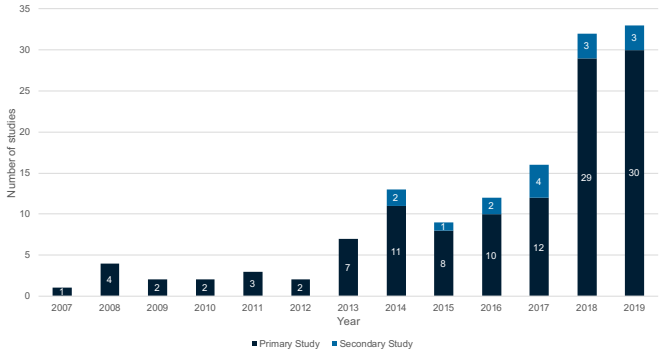


Figure 26: Distribution of used research data types over time

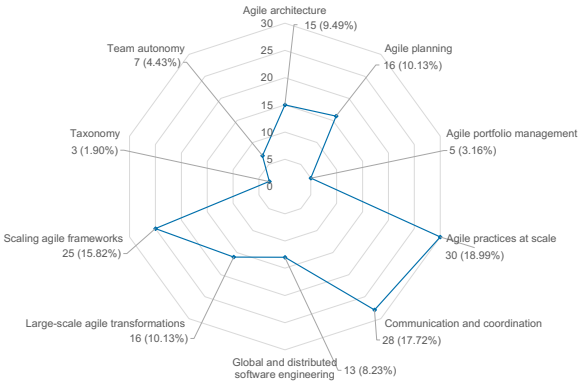


Figure 27: Distribution of the studies in the different research streams

increase in research interest in all research streams (except *Taxonomy*). The *Scaling agile frameworks* research stream shows a steady research interest between 2016 and 2019, with a peak in 2019, accounting for almost one-third of all published studies in 2019. This observation is congruent with Uludağ et al. [101] stating that many scaling frameworks were created and published between 2011 and 2018, confirming the increasing industry interest in scaling frameworks and sparking a growing interest in analyzing the adoption of these frameworks. Furthermore, 56.25% of all studies published between 2018 and 2019 were related to *Agile practices at scale*, *Communication and coordination*, and *Scaling agile frameworks*, while the remaining seven streams accounted for 43.75% of all studies.

Following Kitchenham et al. [58], we derive a research agenda that scholars can use. Figure 29 shows the number of research questions identified for each research stream. We identified 81 research questions that were mapped to the respective streams (see Table 12). Most research questions were identified in the *Communication and coordination*, *Agile architecture*, and *Agile planning* research streams.

Next, we discuss the research streams and gaps reported in the selected studies.

4.4.2. Agile architecture

Agile methods imply that architecture emerges from the system rather than being imposed by some direct structuring force [10]. The practice of emergent design is effective at the team level but insufficient when agile methods are applied at a larger scale, as large-scale agility is enabled by architecture, and vice versa [64], [S106]. The topic of agile architecture is gaining attraction among agilists and researchers (cf. [41, 73, 91]). The *Agile architecture* research stream aims to address the questions of how companies can combine the at first glance contradictory topics of agility and architecture to build complex products and how agile teams can be jointly architecturally aligned at the enterprise level. As the topic of agile architecture is multi-faceted, we identified six sub-topics in the *Agile architecture* stream. The first two sub-topics address agile architecture research at different organizational levels, e.g., at product level, i.e., software architecture, and enterprise level, i.e., enterprise architecture (cf. [S25], [S81], [S102]). Due to the multitudinous-

Table 10: Top 10% of publications ranked by number of citations according to Google Scholar (data collected on 2019-12-31)

Study	Title	Authors	Publication type	Year	No. of citations	Citations per year
[S22]	Challenges and success factors for large-scale agile transformations: a systematic literature review	Kim Dikert, Maria Paasivaara, Casper Lassenius	Journal	2016	273	68
[S6]	A framework to support the evaluation, adoption and improvement of agile methods in practice	Asif Qumer, Brian Henderson-Sellers	Journal	2008	256	21
[S1]	A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case	Kai Petersen, Claes Wohlin	Journal	2009	229	21
[S2]	Agile methods rapidly replacing traditional methods at nokia: a survey of opinions on agile transformation	Maarit Laanti, Outi Salo, Pekka Abrahamsson	Journal	2011	227	25
[S92]	The effect of moving from a plan-driven to an incremental software development approach with agile practices: an industrial case study	Kai Petersen, Claes Wohlin	Journal	2010	166	17
[S112]	Agile portfolio management: an empirical perspective on the practice in use	Christoph J. Stettina, Jeanette Hörz	Journal	2015	126	25
[S34]	Distributed agile development: using scrum in a large project	Maria Paasivaara, Sandra Durasiewicz, Casper Lassenius	Conference	2008	119	10
[S103]	Using scrum in a globally distributed project: a case study	Maria Paasivaara, Sandra Durasiewicz, Casper Lassenius	Journal	2008	101	8
[S3]	Communities of practice in a large distributed agile software development organization – case ericsson	Maria Paasivaara, Casper Lassenius	Journal	2014	98	16
[S53]	Inter-team coordination in large- scale globally distributed scrum: do scrum-of-scrums really work?	Maria Paasivaara, Casper Lassenius, Ville T. Heikkilä	Conference	2012	91	11
[S5]	A case study on benefits and side-effects of agile practices in large-scale requirements engineering	Elizabeth Bjarnason, Krzysztof Wnuk, Björn Regnell	Workshop	2011	87	10
[S24]	Combining agile software projects and large-scale organizational agility	Petri Kettunen, Maarit Laanti	Journal	2008	86	7
[S43]	Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation	Torgeir Dingsøyr, Nils B. Moe, Tor E. Fægri, Eva A. Seim	Journal	2018	83	42

ness of the agile architecture topic, we also identified several sub-topics related to software and enterprise architecture sub-topics. Within the software architecture sub-topic, extant literature aims to describe how software reuse can be facilitated in large-scale agile endeavors (cf. [S25]) and how technical debts can be managed in large-scale agile projects (cf. [S64]). Several studies also aim to clarify the roles and responsibilities of software architects in large-scale agile projects (cf. [S43], [S81], [S104]). In the enterprise architecture sub-topic, we identified two additional sub-topics that aim to answer the questions on how enterprise architecture efforts can be effectively adopted in agile environments (cf. [S11], [S102]) and how enterprise architects can support agile teams (cf. [S57], [S106]). The remaining four sub-topics introduce commonly applied architectural toolboxes, i.e., architecture principles, domain-driven design, models, and patterns (cf. [S39], [S87], [S107], [S110]). Notable results of the *Agile architecture* research stream are:

- Observations indicating that an upfront architecture provides agile teams with a stable working environment contributes to team efficiency and that the role of architects is demanding, requiring continuous coordination with many stakeholders [S43].
- An organizational framework, including roles, teams, and practices, to address challenges related to agile architecting in large-scale agile projects [S107].
- A set of architectural tactics for addressing architectural concerns in large-scale agile projects, e.g., the runway-building tactic, which can be used when architecture is needed that is sufficient to meet near-term needs without causing delays or additional work [S110].

Observations suggesting that an upfront architecture provides agile teams with a stable working environment that likely contributes to team efficiency, and that the role of architects in large-scale agile development is challenging and requires ongoing coordination and negotiation among many stakeholders

Agile methods have been criticized for their lack of focus on architecture [38], assuming that the best architectures emerge from self-organizing teams [2, 10]. For instance, the incremen-

Table 11: Tabular overview of research streams on large-scale agile development

Research stream	Influential studies	Topic level 1 / Study	Topic level 2 / Study	Topic level 3 / Study
Agile architecture	[S43]	Software architecture	Software reuse	[S25]
			Role of software architects	[S43], [S57], [S81], [S104]
			Technical debts	[S64]
		Enterprise architecture	Challenges and success factors	[S11], [S102]
			Role of enterprise architects	[S57], [S106]
		Architecture principles	[S39], [S41], [S56]	-
		Domain-driven design	[S87]	-
		Models	[S107]	-
		Patterns	[S110]	-
Agile planning	[S5], [S43]	Effort estimation	[S37]	-
		Release planning process	[S8], [S27], [S63], [S68]	-
		Challenges and success factors	[S5], [S26], [S69], [S91], [S101], [S125], [S134]	-
		Customer involvement	[S33], [S43], [S98], [S114]	-
		Not specified	[S74]	-
Agile portfolio management	[S112]	Challenges and success factors	[S88], [S112], [S127], [S128]	-
		Patterns	[S15]	-
Agile practices at scale	[S1], [S3], [S24]	Tailoring	[S14], [S18], [S61], [S80], [S90]	-
		Kanban	[S131]	-
		Retrospectives	[S62]	-
		Communities of practices	[S3], [S38], [S84], [S118]	-
		Rapid application development	[S96]	-
		Challenges and success factors	[S35], [S46], [S47], [S78], [S126], [S135]	-
		Models	[S19], [S24], [S32], [S41], [S87], [S133]	-
		Patterns	[S35], [S46]	-
		Not specified	[S1], [S75], [S94], [S111], [S119]	-
Communication and coordination	[S43]	Inter-team coordination	Multiteam systems	[S30], [S51], [S52], [S93]
			Coordination mechanisms	[S20], [S28], [S31], [S36], [S43], [S54], [S70], [S71], [S85], [S97], [S115]
			Team performance	[S48], [S58]
			Challenges and success factors	[S29], [S117]
			Software ecosystems	[S130]
		Knowledge sharing	Knowledge boundaries	[S76]
			Knowledge networks	[S16], [S67], [S83], [S104]
			Not specified	[S44], [S59], [S84]
Global and distributed software engineering	[S34], [S103]	Challenges and success factors	[S10], [S34], [S66], [S72], [S89], [S99], [S103], [S136]	-
		Offshoring and outsourcing	[S14], [S18], [S61], [S80]	-
		Not specified	[S37]	-
Large-scale agile transformations	[S2], [S22], [S92]	Challenges and success factors	[S2], [S4], [S21], [S22], [S40], [S50], [S60], [S65] , [S92], [S100], [S121], [S122], [S123]	-
		Agile mindset	[S46], [S86]	-
		Theory building	[S113]	-
Scaling agile frameworks	[S6], [S53]	Challenges and success factors	[S49]	-
		Large-Scale Scrum	[S42], [S55], [S79]	-
		Scaled Agile Framework	[S13], [S19], [S36], [S71], [S95], [S115], [S116], [S124], [S129], [S132]	-
		Scrum-of-Scrums	[S53]	-
		Spotify Model	[S12], [S45], [S84]	-
		Agile Software Solution Framework	[S6]	-
		Disciplined Agile Delivery	[S40]	-
		Framework comparisons	[S9], [S17], [S57], [S77], [S108]	-
Taxonomy		Not specified	[S7], [S73], [S105]	-
Team autonomy		Challenges and success factors	[S23], [S39], [S109], [S116]	-
		Governance	[S16], [S82], [S120]	-

Table 12: Tabular overview of open research questions for large-scale agile development

Agile architecture

1. How can technical debts be managed and minimized in large-scale agile projects?
2. How is the role of enterprise architects practiced in large-scale agile development?
3. How do architects collaborate with agile teams in large-scale agile development?
4. How can architecture drive large-scale agile transformations?
5. How can the decision-making power between architects and agile teams be balanced?
6. How can software architecture support the coordination of agile teams?
7. How can coordination mechanisms improve architecture sharing at intra- and inter-team level?
8. How can emergent and intentional architecture be balanced?
9. How can the compliance of agile teams with architecture principles automatically determined?
10. What is the effect of applying architecture principles on the outcome of large-scale agile transformations?
11. What are good practices for addressing challenges related to the establishment of architecture principles?
12. Which typical challenges do architects face in large-scale agile development?

Agile planning

1. How do organizations that have adopted agile methods implement release planning?
2. How do product owners and customers collaborate with developers in large-scale agile projects?
3. What legal limitations exist in contracts that reduce agility in large scale projects?
4. How can technical dependencies between agile teams be minimized?
5. How can the prioritization between functional and non-functional requirements be balanced in large-scale agile projects?
6. What are good contracting models for organizations with external customers?
7. What are typical requirements engineering challenges in large-scale agile development?
8. Which ceremonies can be used to improve the release planning process?
9. What are factors that impact the accuracy of effort estimations in large scale agile projects?
10. How can customer representatives and agile teams be aligned in large-scale agile projects?
11. How can high-level planning elements be incorporated in agile daily routines of large-scale agile projects?

Agile portfolio management

1. What are best practices in agile portfolio management?
2. How is portfolio management interrelated with other governance functions in an agile context?
3. How can traditional portfolio management techniques be applied in an agile environment?
4. How is the new role of project managers practiced in large-scale agile development?
5. How do agile methods affect program and portfolio management?
6. How to enable the strategic alignment and management of agile project portfolios?

Agile practices at scale

1. What are challenges, benefits, and success factors of scaling agile practices in organizations?
2. What are recurring concerns and good practices of typical stakeholders in large-scale agile development?
3. What are challenges, benefits, and success factors of establishing communities of practice in large-scale agile projects?
4. How can the on-boarding of new agile team members be facilitated in large-scale agile projects?
5. What is the impact of applying agile practices to the overall performance of the organization?
6. What are appropriate metrics to monitor the progress of agile teams and to support transparency in large-scale agile projects?
7. How can agile practices be scaled in organizations from the public sector?
8. How can continuous improvement at intra- and inter-team level be facilitated?
9. Which issues arise when are retrospectives are organized at inter-team level?
10. How can agile practices be adapted to meet inter-organizational needs?

Communication and coordination

1. How can coordination mechanisms be applied effectively in large-scale agile development?
2. How can effective knowledge networks be created in large-scale agile projects?
3. Which tools can be used to support inter-team coordination in large-scale agile projects?
4. How can the number of meetings in large-scale agile projects be reduced?
5. How can meetings in large-scale projects be designed to increase the effectiveness of coordination?
6. How is intra-team coordination affected by increased focus on inter-team coordination or vice versa?
7. Based from a multi-team perspective, how is coordination in large-scale agile development performed?
8. What are effective organizational structures and collaboration models in large projects?
9. Which effect do cultural differences have on inter-team coordination large-scale agile projects?
10. How can focused work and knowledge be balanced in large-scale agile projects?
11. How does co-location of agile teams affect knowledge sharing in large-scale agile projects?
12. Which challenges are caused by inter-team dependencies within large-scale agile projects?
13. How can daily stand-up meetings be organized in a way that they enable inter-team coordination?

Global and distributed software engineering

1. What are challenges, benefits, and success factors of applying agile practices in distributed projects?
2. How can virtual agile teams be supported in distributed software development projects?
3. How is knowledge sharing performed in distributed large-scale agile projects?
4. Which human related factors can positively affect globally distributed software projects?
5. How is frequent communication in distributed projects enabled to overcome the challenges of distance?

Large-scale agile transformations

1. What are challenges, benefits, and success factors of performing large-scale agile transformations?
2. How can non-agile units be integrated with agile organizational units to support agile transformations?
3. What are reasons and consequences of conducting large-scale agile transformations on the organizations?
4. How can hierarchical and organizational structures be reduced to facilitate large- scale agile transformation?
5. How are agile structures adopted in business units that are not engaged in IT development or delivery?
6. How can local optimization of agile teams be aligned with the enterprise strategy?
7. Which KPIs exist to measure the enterprise agility?
8. How are agile methods adopted at large-scale in highly regulated environments?
9. How do agile teams adopt common values within large-scale agile transformations?

Scaling agile frameworks

1. Which scaling agile frameworks are used in organizations and what are their benefits and challenges?
2. How can scaling agile frameworks be selected that are suitable for specific contexts?
3. How is the Scaled Agile Framework adopted in organizations and what are respected challenges and risks when adopting it?
4. How and when should be scaling agile frameworks used in large-scale agile projects?
5. How are scaling agile frameworks tailored to meet the needs of the organizations in which they are adopted?
6. How is the Large-Scale Scrum framework adopted in different types of organizations?
7. Which performance improvements can be observed when adopting scaling agile frameworks?
8. How can agile release trains and value streams be formed in complex organizations?

Taxonomy

1. How can agile in the large be conceptualized besides the dimension of number of teams?

Team autonomy

1. How can team autonomy in large-scale agile development be increased?
2. How can inter-team coordination and team autonomy be balanced in large-scale agile projects?
3. What are effective intra- and inter-team coordination mechanisms for autonomous agile teams?
4. How can autonomous teams be designed, supported, and coached?
5. What are barriers to team autonomy in large-scale agile development?
6. How do governance structures influence team autonomy in large-scale agile development?

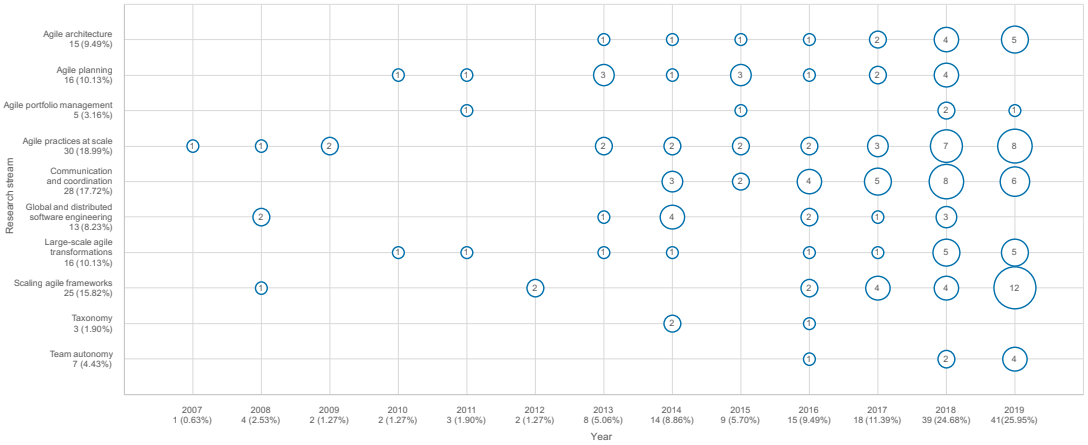


Figure 28: Number of studies per research stream over time

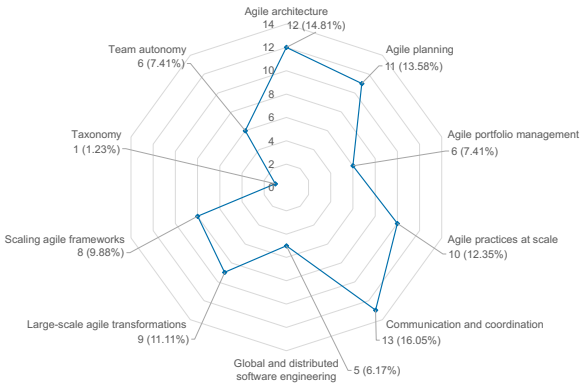


Figure 29: Distribution of research questions in the different research streams

tal design practice of XP asserts that architecture emerges in daily design [10, 13]. Apart from verbal discussions of design decisions, Scrum does not emphasize architecture-related practices as the architecture of a single-project application can always be re-factored and repackaged for a higher level of reuse [10]. There is growing interest from academics and practitioners who wish to combine the two concepts of agility and architecture as some degree of architectural planning and governance becomes increasingly crucial for large-scale agile projects [64]. Due to the existing ambiguity on agile architecture in large-scale agile development, researchers have formulated 12 open research questions on this topic. The first two research questions aim to study the collaboration between architects and agile teams (cf. [W5], [S81], [S87]) and to analyze the tension between architects with decision-making power and self-organizing agile teams (cf. [W5], [S104]). Three studies (cf. [W2], [W6], [S64]) mention the issue of managing technical debts as a research question.

4.4.3. Agile planning

Many large agile organizations struggle to implement efficient requirements management processes. Moreover, research on agile planning practices in large organizations is scarce [47], [S2]. The *Agile planning* research stream aims to provide scientific evidence on how large-scale agile projects and organizations perform agile planning activities. We identified four sub-topics in this stream. While a lot of research has covered effort estimation in software development projects [52], little research has been conducted on effort estimation in large-scale distributed projects [S22]. Thus, the first sub-topic aims to investigate how large-scale agile projects perform effort estimation (cf. [S37]). Given the importance of release planning to the success of a development project and the lack of solid empirical evidence in the research of release planning in large-scale agile organizations [S68], the second sub-topic aims to describe how these organizations perform release planning. Similar to other research streams, the third sub-topic aims to report on challenges and success factors in adopting agile planning practices (cf. [S5], [S101], [S134]). While agile methods strongly emphasize customer involvement, companies struggle to meet the needs of customers as very large programs encompass many requirements, stakeholders and developers [34], [S33]. Hence, the last sub-topic aims to reveal how large-scale agile projects can actively involve customers (cf. [S33], [S43], [S98]). Notable results pertaining the *Agile planning* research stream are:

- Observations showing that incorporating agile practices into requirements engineering can overcome existing challenges, e.g., communication gaps and over-scoping, but can also introduce new challenges, e.g., balancing agility and stability and ensuring sufficient competencies in cross-functional teams (see [S5]).
- Observations indicating that the scale of a large-scale agile project, e.g., the number of sites, the number of stakeholders, the size and complexity of the legacy code, and the coordination between different types of stakeholders make estimation and planning very challenging (see [S37]).
- A list of ten benefits, e.g., improved communication, transparency, and dependency management, and nine challenges, e.g., lack of preparation and prioritization of requirements, unrealistic schedules, and inadequate architectural planning, when using a release iteration planning method in large-scale agile projects (see [S68]).

We identified 11 research questions for future studies. Al-

though release planning is a critical success factor in agile software projects [26], there is little research on large-scale agile release planning [S27]. Researchers call for studying how companies that have adopted agile methods implement release planning (cf. [W1], [S8], [S27]). They suggest analyzing how companies can use agile ceremonies in the release planning process and how companies can incorporate high-level planning elements into daily routines of large-scale agile projects (cf. [S26], [S51]). As large-scale agile programs often have a large number of stakeholders and users, and their needs have to be communicated to a large number of developers, customer engagement can be a major challenge [S43]. Thus, scientists suggest further research on customer-developer collaboration in large-scale agile projects (cf. [W1], [W2]). They also call for research on identifying appropriate contracting models for companies with external customers (cf. [W6]) and how companies can facilitate alignment between customer representatives and agile teams in large-scale agile projects (cf. [S43]). Researchers should examine legal constraints in contracts limiting the agility of large-scale projects (cf. [W1], [W2]).

4.4.4. Agile portfolio management

While the concept of portfolio management is not new [S88] and is established in the traditional project management literature, the iterative nature of agile methods poses new challenges to the current management practice [S112], e.g., the need for portfolio management to be able to respond quickly to changes while connecting agile teams to strategy [S15]. The *Agile portfolio management* research stream deals with the question of how companies can adapt their traditional portfolio management approaches to agile environments. This stream can be divided into two sub-topics, namely identifying challenges and success factors of agile portfolio management approaches (cf. [S88], [S112], [S127]) and creating patterns for agile portfolio management (cf. [S15]). Stettina and Hörz [S112] provide an overview of portfolio management practices of 14 large European companies affected by agile methods and describe the challenges, implications, and benefits of agile methods on portfolio management practices. Horlach et al. [S15] propose four design goals for an effective agile portfolio management and six design principles for achieving these goals. Interesting findings of the *Agile portfolio management* research stream include:

- Observations indicating that implementing a structured portfolio management process can help reduce time-to-market, improve project visibility, and coordinate the work of agile teams (see [S88]).
- Observations suggesting that most challenges associated with agile portfolio management are process-related and that an alignment with the customer and involvement of the business is beneficial (see [S112]).
- Formulation of 16 propositions for the effective management of agile projects portfolios, e.g., that portfolio managers need to find the right balance between control and autonomy in agile projects, and that agile project portfolios require mechanisms for simple, fast, and collective decision-making (see [S127]).

Within the *Agile portfolio management* research stream, we identified six open research questions. As existing best practices for portfolio management are more suited for stable environments [S15], two studies (cf. [W6], [S15]) suggest researchers identifying best practices for agile portfolio management. Hobbs and Petit [S111] suggest further research on the impact of adopting large-scale agile methods on portfolio management. Rautiainen et al. [S88] recommend exploring how companies can apply traditional portfolio management techniques in agile environments.

4.4.5. Agile practices at scale

Adopting and scaling agile practices outside of their ideal context requires rethinking the original underlying assumptions of agile practices [S73]. The *Agile practices at scale* research stream aims to explore how companies tailor genuine agile practices to fit large-scale projects. We identified eight sub-topics in this research stream. The first sub-topic mainly analyzes how companies tailor Scrum-related roles and artifacts in large projects (cf. [S61], [S80], [S90]). The next four sub-topics describe the application of different agile and lean practices in large agile projects, namely Kanban (cf. [S131]), Retrospectives (cf. [S62]), Community of Practices (cf. [S3], [S84], [S118]), and Rapid Application Development (cf. [S96]). The sixth sub-topic, like other research streams, deals with identifying challenges and success factors for applying agile practices in large-scale projects (cf. [S47], [S78], [S135]). The seventh sub-topic mainly generalizes and conceptualizes observations from case studies in the form of models related to applying agile practices in large-scale projects (cf. [S32], [S41], [S133]). Inspired by existing pattern languages on agile software development, the eighth sub-topic aims to provide best practices for large-scale agile endeavors (cf. [S35], [S46]). Notable findings of the *Agile practices at scale* research stream are:

- Observations showing that adopting agile practices in large-scale projects brings several benefits, e.g., increased control and transparency over the project and better learning and understanding through direct communication, but also raises new issues, e.g., administrative overhead and coordination, and limited focus on architecture (see [S1]).
- Observations pointing that organizational culture, previous agile experience, and management support are key success factors in scaling agile practices and that resistance to change, an overly aggressive time-frame for adoption, and integration with existing non-agile business processes are critical challenges in scaling agile methods (see [S78]).
- A study showing that medium and large software projects using agile methods perform better on average than projects using non-agile methods and that agile methods are more likely to succeed compared to traditional methods as project size increases (see [S119]).

The use of agile methods on large-scale projects brings unprecedented challenges [S43]. We identified 10 related research questions, of which the unveiling the challenges, benefits, and success factors of scaling agile practices in organizations was the most frequently stated (cf. [S2], [S78], [S125]). Although

several pattern languages documenting best practices for agile projects have been published in agile software development research (cf. [15, 16, 27]), the body of knowledge about best practices in large-scale agile development is still emerging. Several studies suggest identifying best practices for large-scale agile endeavors (cf. [S1], [S35], [S78]). Given the importance of communities of practices for knowledge sharing, inter-team coordination, and technical work [63], [S3], several researchers recommend future studies to investigate the challenges, benefits, and success factors of establishing communities of practices in large-scale agile projects (cf. [W3], [W6], [S3]). Since onboarding newcomers in large-scale agile projects might be challenging [W3], three studies propose as future work to investigate how companies can facilitate the onboarding process of new agile team members in these projects (cf. [W3], [W6], [W9]). Several studies suggest identifying metrics to quantify the impact of applying agile practices on overall organizational performance and to monitor the progress of agile teams in large-scale agile projects (cf. [W2], [S26], [S63]).

4.4.6. Communication and coordination

Coordinating work is critical when managing large projects with multiple teams [S97]. This circumstance is also true for large-scale agile projects, as work is performed by many developers and development teams simultaneously, and the frequent and iterative delivery of results requires work and knowledge coordination at different levels [S43, S54]. Achieving effective coordination in large-scale agile projects is difficult due to the complexity of these projects [S54]. The *Communication and coordination* research stream tackles the issue of how agile teams can effectively coordinate and communicate with each other. The *Communication and coordination* stream covers multiple topic levels consisting of two sub-topics, namely inter-team coordination and knowledge sharing, and another seven subordinate sub-topics. Since coordination between teams is critical in managing large-scale endeavors [S31], the inter-team coordination sub-topic mainly aims to identify and describe different coordination mechanisms and coordination modes for aligning agile teams (cf. [S28], [S31] [S54]). Within the inter-team coordination sub-topic, we identified five additional sub-topics that aim (i) to advance the conceptual understanding of inter-team coordination through the lens of multi-team systems (cf. [S30], [S52], [S93]), (ii) describe different mechanisms and modes for coordinating agile teams (cf. [S31], [S85], [S97]), (iii) reveal factors that influence the performance of agile teams (cf. [S48], [S58]), (iv) identify challenges and success factors related to the coordination of agile teams (cf. [S29], [S117]), and (v) describe the adoption of agile methods beyond organizational boundaries (cf. [S130]). Since resource availability in a team's knowledge network and the effective knowledge coordination between agile teams become paramount in large-scale agile projects [S83], the knowledge sharing sub-topic mainly aims to describe how agile teams can share their knowledge and expertise with other teams. We identified two sub-topics in the knowledge sharing sub-topic that aim to answer the questions on how companies tailor agile practices to enable scaling across different knowledge boundaries (cf. [S76]) and how companies can build effective knowledge networks in large-scale agile projects (cf. [S16], [S67], [S83]). Notable results of the *Communication and coordination* research stream are:

- Observations suggesting that the combination of traditional planning at an inter-team level and agile development at a team level leads to ineffective coordination in large-scale agile development, as a lack of dependency awareness is hampered by misaligned planning activities at the team and inter-team levels (see [S29]).
- Observations indicating that group mode coordination is central for achieving effective inter-team coordination in large-scale agile projects, using scheduled and unscheduled meetings, e.g., product owner meetings and spontaneous discussions in the open work area (see [S83]).
- Observations pointing that networking is essential in solving complex tasks in large-scale agile projects and that several mechanisms can support the creation of knowledge networks, e.g., introducing formal technical experts and facilitating communities of practice (see [S97]).

Due to the high number of articles in the *Communication and coordination* research stream, we identified 13 research questions that were most frequently for a given stream. Because the coordination of many agile teams is a key challenge [S28], several researchers emphasize further research to investigate how companies can design and apply coordination mechanisms for increasing the effectiveness of coordination (cf. [S18], [S28], [S70]). They also call for research to analyze how companies can reduce the number of meetings in large-scale agile projects (cf. [W5], [S70]) and identify tools that companies can use for inter-team coordination (cf. [W6], [W8], [S70]).

4.4.7. Global and distributed software engineering

Given the benefits to both customer and vendor companies in terms of low cost, early product delivery, and high-quality products, companies are increasingly deploying virtual teams that operating across geographical boundaries to develop software [72, 75]. The competitive advantages and business profits of agile methods motivate companies to adopt agile methods in large, globally distributed projects [S89]. Agile methods are more difficult to scale in distributed projects due to additional challenges, e.g., communication and coordination issues, cultural differences, and temporal issues [S22], [S103]. The *Global and distributed software engineering* research stream deals with how companies can use agile methods in large, globally distributed projects. Studies mainly focus on two sub-topics, namely identifying challenges and success factors in adopting agile methods in large, globally distributed projects (cf. [S72], [S89], [S103]) and the tailoring of agile methods to meet the needs of these projects (cf. [S18], [S61], [S80]). As an example for the first sub-topic, Shameem et al. [S89] describe critical factors that positively impact the adoption of agile methods in large, globally distributed projects based on a systematic literature review. Related to the second sub-topic, Bass [S14] uses 46 interviews with eight companies to investigate how companies adapt the role of the product owner to the needs of large, geographically distributed software projects.

Noteworthy findings related to the *Global and distributed software engineering* research stream include:

- A list of 11 challenges identified in large, globally distributed agile projects, of which 6 were evaluated as critical: lack of requirements analysis, customer involvement, communication, roles and responsibilities, management commitments, and knowledge sharing (see [S10]).
- A classification model for large, globally distributed agile projects contrasting solution focused development and execution focused development (see [S61]).
- Observations indicating that adopting agile practices based on Scrum in a large, globally distributed agile project helps mitigate the biggest problem of large, globally distributed projects, namely communication, by creating frequent possibilities to communicate across distributed sites, e.g., through Daily Scrum meetings (see [S103]).

We identified five open research questions in the *Global and distributed software engineering* stream. As many companies scale agile methods to distributed environments due to fast development rates and low development costs [S72], identifying challenges, benefits, and success factors when scaling agile to distributed organizations was the most frequently cited research question (cf. [W1], [S72], [S103]).

4.4.8. Large-scale agile transformations

Companies are striving to become agile to respond to dynamic environments and sustain their survival. As a result, many companies are extensively introducing agile methods leading to large-scale agile transformations [S4], [S21], [S22]. These transformations entail new managerial challenges, e.g., lack of top management engagement [S21], skepticism towards the new way of working [S22] or establishing an enterprise-wide agile culture and mindset [S46]. The *Large-scale agile transformations* research stream aims to shed light on how companies undergo these transformations to meet the imperatives of agile companies. We identified three sub-topics in this research stream. Like other streams, the first sub-topic deals with identifying a set of observed success factors and challenges when companies undertake large-scale agile transformations (cf. [S2], [S4], [S100]. As these transformations represent large episodic change processes that have a large impact on the employees [S21] and changing the peoples' mindset is more difficult than teaching new practices [S46], the second sub-topic discusses various ways in which companies can establish an agile mindset and define common values among employees (cf. [S46], [S86]). The third sub-topic takes a more theoretical stance and aims to theorize the process behind large-scale agile transformations (cf. [S113]). Fuchs and Hess [S21] provide an example for the first sub-topic and captures the interplay of challenges, coping, and scaling actions in the execution of large-scale agile transformations through socio-technical systems theory. Relating to the second sub-topic, Paasivaara et al. [S86] report how Ericsson established value workshops to align different sites and teams as part of its large-scale agile transformation. Carroll and Conboy [S113] provide an example for the third sub-topic, applying normalization process theory to examine the normalization of these transformations. Notable results of the *Large-scale agile transformations* research stream are:

- A survey indicating that while a large-scale agile transformation poses new challenges, e.g., agile deployment and adaptation of agile methods to fit the organization and requirements management and planning in a flexible and iterative manner, it also brings several benefits, e.g., higher satisfaction, a feeling of effectiveness, and increased transparency and autonomy, which is why most respondents do not want to return to the old way of working (see [S2]).
- A list of 35 challenges, e.g., agile difficult to implement and integrating non-development functions, and 29 success factors, e.g., management support and choosing and customizing the agile model, were reported for large-scale agile transformations (see [S22]).
- Observations stating introducing agile practices in an organization that has been working in a plan-driven way leads to several improvements, e.g., increased release frequency and better reflection of the customers' needs, but also raises several challenges, e.g., support for coordinating a high number of teams and integration of release projects in the overall development process (see [S92]).

As the *Large-scale agile transformations* research stream has flourished in recent years, especially in the last two years, several studies formulated important research questions as avenues for future research. The most frequently asked research question aims to identify challenges, benefits, and success factors in conducting large-scale agile transformations (cf. [S24], [S45], [S100]). As "*agile breaks everything*" and adopting agile practices may have far-reaching implications for companies [62], [S4], several researchers call for examining the reasons for performing large-scale agile transformations and their impact on companies (cf. [S5], [S45], [S65]). Since companies often encounter challenges when agile units need to collaborate with non-agile departments [62], some studies propose to investigate how companies have overcome these challenges from a longitudinal perspective (cf. [W7], [S4], [S45]). From an integrational perspective, three studies suggest further research on how companies can adopt agile structures in business units that are not engaged in software development or delivery (cf. [W4], [W6], [S45]). Hierarchical control and bureaucracy mechanisms can act as barriers for the successful performance of large-scale agile transformations [S122], researchers should explore how companies can dissolve their hierarchical structures to facilitate these transformations (cf. [W4], [W6], [W7]).

4.4.9. Scaling agile frameworks

Some custodians of existing agile methods and practitioners have created several scaling frameworks that claim to provide off-the-shelf solutions for solving issues related to the large-scale adoption of agile methods [101], [S43], [S113]. As there is a increasing interest in adopting scaling frameworks from a practical perspective [31], there is also a growing academic interest in studying the adoption of these frameworks within the *Scaling agile frameworks* research stream. We identified eight sub-topics in this stream. As empirical evidence on the

adoption of scaling frameworks is still very much in its infancy [S49], the first sub-topic aims to highlight challenges and recommendations for companies seeking to adopt scaling frameworks (cf. [S49]). The other six sub-topics mainly analyze how popular scaling frameworks are adopted (cf. [S12], [S36], [S40]). The last sub-topic aims to provide a comparison of these frameworks (cf. [S9], [S17], [S77]). Conboy and Carroll [S49] provide an example for the first sub-topic and unveils nine challenges associated with implementing scaling frameworks and a set of recommendations to address these challenges adequately. Relating to the six sub-topics analyzing the adoption of popular frameworks, Petit et al. [S12] present a case study of how a large company adapted Spotify practices to promote the effectiveness of team autonomy in a mission-critical project. Lal and Clear [S40] provide another example, presenting a case study on how a global software vendor transitioned to an agile company by adopting DAD. Related to the last sub-topic, Alqudah and Razali [S9] compare six scaling frameworks, e.g., DAD, LeSS, and SAFe, based on various criteria, e.g., team size and available training. Noteworthy results of the *Scaling agile frameworks* research stream are:

- A maturity model consisting of levels, principles, and practices for defining a roadmap for SAFe adoption and assessing the level of adoption (see [S19]).
- A list of nine challenges, e.g., comparing scaling frameworks and top-down vs. bottom-up implementation, and a set of recommendations to overcome these challenges, e.g., using a small number of metrics to compare frameworks and determining whether a framework promotes a top-down or bottom-up implementation approach, when implementing scaling frameworks (see [S49]).
- A survey indicating that the SAFe adoption leads to several benefits, e.g., improved transparency, cooperation, and cadence, but also entails several obstacles, e.g., old mindset and culture, SAFe has not correctly fitted to the organization, and missing fluency when using SAFe (see [S129]).

The selected studies within the *Scaling agile frameworks* research stream named eight research questions for further investigation by researchers. The most frequently stated research question on scaling agile frameworks deals with the observation of adopting specific scaling frameworks in companies and the associated benefits and challenges (cf. [W6], [S40], [S60]). As scaling frameworks are often not selected systematically but merely based on the popularity of the framework or recommendation of consultants, future work should identify contextual factors and comparison criteria that companies can use to select scaling frameworks systematically (cf. [W3], [S79], [S92]). Since many companies also adapt scaling frameworks to their organizations, two studies (cf. [S22], [S60]) call for future research on how companies tailor scaling frameworks to meet their needs. There is also a call for research on the two most widely adopted scaling frameworks, namely SAFe and LeSS, to study their adoption in companies (cf. [S19], [S55], [S124]).

4.4.10. Taxonomy

The *Taxonomy* research stream deals with providing more conceptual clarity related to large-scale agile development, as well as the scale and implications for scalability of agile methods. Power [S7] explores when scaling approaches are appropriate in large organizations and discusses three contexts for agility at scale. Rolland et al. [S73] take a more theoretical stance to clarify the meaning of large-scale agile development by examining its underlying assumptions in existing studies. Dingsøyr et al. [S105] provide a taxonomy for characterizing large-scale agile projects based on the number of agile teams. Noteworthy results of the *Taxonomy* research stream are:

- Three contexts for agility at scale: (i) being agile in a team inside a large organization, (ii) using agile approaches in a large development effort inside a large organization, (iii) and the large organization being itself agile (see [S7]).
- A comparison of 10 prevailing assumptions in existing studies with a set of alternative assumptions better suited to the characteristics of large-scale agile projects, e.g., collective code ownership vs. transferring, translating, and transforming knowledge across boundaries (see [S73]).
- A taxonomy of scale, consisting of three categories: (i) small-scale (1 team), (ii) large-scale (2–9 teams), and very large-scale ($\geqslant$10 teams) (see [S105]).

We identified one research question in the *Taxonomy* stream. Two articles (cf. [S73], [S105]) suggest further research verifying the current taxonomy of scaling, which is currently merely based on the number of teams involved in large-scale agile projects [S105], and identifying other adequate scaling dimensions for classifying the scale of agile development projects.

4.4.11. Team autonomy

In large-scale agile development, the effective functioning of team autonomy is challenged as a certain amount of autonomy has to be sacrified to reach consensus on common standards and align work with other teams [S109], [S116]. The *Team autonomy* research stream is primarily concerned with how complex organizations affect team autonomy and how they can strike a balance between self-organizing teams focused on their own goals and those of the broader organization. We identified two sub-topics in *Team autonomy* stream related to identifying challenges and success factors in establishing team autonomy in large-scale agile endeavors (cf. [S23], [S109], [S116]) and in the context of the interplay between governing and autonomizing agile teams (cf. [S16], [S82], [S120]). For instance, Moe et al. [S23] analyze large-scale projects and presents barriers that reduce the extent of team autonomy. Concerning the second sub-topic, Šāblis and Šmite [S16] investigate the interplay between governance and team autonomy by identifying governance roles that support teams. Interesting findings related to the *Team autonomy* research stream include:

- Observations indicating that there are two barriers to team autonomy in large-scale agile development: overarching goals that are often set by management without team involvement and organizational dependencies that result in teams having to manage additional tasks (see [S23]).

- Observations suggesting that reliance on resources outside of the large-scale agile program, e.g., shared IT resources, creates external dependencies and reduces the autonomy of agile teams (see [S109]).
- A model for assessing the autonomy of agile teams in large-scale agile development (see [S120]).

We identified six research questions in the *Team autonomy* research stream. The most frequently cited research questions relate to how to improve team autonomy in large-scale agile endeavors (cf. [W4], [S16], [S29]) and how to balance coordination between agile teams and their autonomy in large-scale agile projects (cf. [W8], [S16], [S43]), as the need for coordination and control can restrict a team's autonomy [S16].

5. Discussion

In this section, we discuss our general observations on the start-of-the-art on large-scale agile development.

Reflection on the research of the past 13 years. Since the first publication in 2007, researchers worldwide have lavished attention to study the application of agile methods in large projects and organizations. While the number of published studies started to accelerate in 2013, the academic interest has been steadily growing. The maturation of the research field can be seen both in the increasing number of published studies, as well as in increasing number of articles in top journals. While the 52 included papers from the most prominent systematic literature review on large-scale agile development by Dikert et al. [S22] comprised mostly experience reports indicating a lack of sound academic research, the 136 included studies in this study covered only research papers excluding personal opinion and experience papers from practitioners and scientists signaling a tremendous shift of the scientific foundation of large-scale agile development. We believe that this academic traction is sparked by the omnipresent industrial relevance of the topic demanding scientific assistance by researchers (cf. [37], [S22]). While almost 60% of the papers represent case studies mainly being exploratory and less theoretical, they do not pay enough attention to establish theoretical underpinnings as similar to studies on agile software development [34]. Analogous to agile software development and agreeing with Dingsøyr et al. [34], we believe that the area of large-scale agile development can mature as a research discipline only if adequate efforts are made to provide a solid theoretical scaffold.

Current structure of the research landscape. Research on large-scale agile development has been conducted mainly empirically and qualitatively to describe and explain how companies adopt large-scale agile methods. Consequently, extant research has been dominated by evaluation research assessing the large-scale adoption of agile methods and deriving a set of lessons learned instead of creating solution artifacts in the form of models, frameworks/methods, and tools. Like Batra [12], we can further observe an apparent lack of quantitative studies using surveys as data collection instruments to provide quantitative investigations and assessments, e.g., quantitatively assessing the strengths and weaknesses of scaling frameworks or performance of agile teams in large agile multi-team settings.

Outlook for future research undertakings. We encountered some intriguing observations, which is why we believe that large-scale agile development will continue to be a relevant topic in practice as well as in academia. An analysis of the past State of Agile surveys conducted by Digital.ai (cf. [29–31]) reveals an ever-increasing adoption of agile methods in organizations, including the adoption of scaling agile frameworks. According to Digital.ai [31], this trend will likely continue and also intensify, especially concerning the more significant expansion and scaling of agile methods beyond software development, namely across the whole company. Parallel to this development, the number of scaling agile frameworks proposed by software practitioners is still flourishing and likely to further grow as the creators of these frameworks feel committed to develop their frameworks further [101]. The number of published studies in the last years (see Figure 4.2) and the increasing interest of various publication venues (see Appendix A) from different research domains (see Figure 17) indicate that the growing industrial interest in large-scale agile development is backed by a growing scientific interest across diverse research communities. Although researchers made considerable efforts to close research gaps in various research streams of large-scale agile development, a high number of open research questions (see Table 12) are still waiting to be addressed by researchers.

Contemplation of the current phenomenon. Although the early advice from the agile community was that scaling agile methods to larger projects and organizations is "*probably the last thing anyone would want to do*" [87], and the advice from several fields is to reduce the size of software projects as much as possible [9], why are companies still trying to adopt agile methods outside of their sweet spot in larger projects and organizations? One plausible explanation is that solutions often demand too much work for a single team, or that new solutions are so complex or so dependent on existing systems that it is considered inefficient or impractical to split the development efforts into small projects, making agile methods a way to reduce risk at scale while also facilitating innovation [37]. Despite the challenges of large-scale adoption of agile methods, we observe that the idea permeates almost every continent and industry sector. We revealed more than 150 companies distributed over the globe across various sectors make the use of agile methods in larger projects and organizations (see Section 4.1.1). Our findings and the survey results of Digital.ai [31] indicate that, regardless of their organizational size, companies are adopting agile methods at scale. However, our results show that most of the adopting companies have more than 5,000 employees, accounting for almost 70% of all identified case companies with stated size, indicating that this phenomenon is probably more relevant for large companies than for small companies.

New emerging research themes. Early research on large-scale agile development started with contributions related to the *Agile practices at scale*, *Global and distributed software engineering*, and *Scaling agile frameworks* research streams (see Section 4.4.1). Themes recently receiving more focus include *team autonomy* and *large-scale agile transformations*. For instance, as large-scale agile transformations can be characterized

as episodic organizational change processes [S21], there is a need to conduct longitudinal case studies that accompany these transformations to investigate their long-lasting effects.

6. Threats to validity

Although we employed a rigorous study design and paid particular attention to the selection and analysis of published studies, our study has some limitations. The results of this systematic mapping study may be affected by various threats to validity, which are (i) study search incompleteness, (ii) study selection bias, (iii) study distribution imbalance, and (iv) data extraction inaccuracy, which we will discuss in the following.

6.1. Incompleteness of study search

There may be some relevant publications that we did not retrieve by our search, which may affect the completeness of our study. To mitigate this risk, we searched the most common electronic databases in which a large number of journals as well as conference and workshop proceedings in the fields of software engineering and information systems are indexed. We also performed a preliminary search before the main search to improve the correctness and completeness of the search results. These measures reduced the probability of missing relevant studies.

6.2. Bias on study selection

The selection of relevant studies largely depends on the personal knowledge of the researchers, which may lead to a bias in the results of the study selection. To mitigate this bias, we created a set of selection criteria (see Section 3.2.2). As the researchers of this study may have different understandings of the selection criteria, we conducted a preliminary search before the main search to ensure that the researchers had a consistent understanding of the selection criteria. Two reviewers also performed the study selection process in parallel and independently and then discussed and resolved any conflicts between their results to mitigate personal bias in study selection. A potential bias may arise from excluding the grey literature, e.g., white papers, technical reports, or editorials. This potential bias is inherent to our study design and did not significantly impact our study. We wanted to focus exclusively on state of the art presented in high-quality scientific papers that have undergone rigorous peer-reviewed publication processes.

6.3. Imbalance of study distribution

Around one-third of the selected publications come from the proceedings of the International Conference on Agile Software Development and International Workshop on Large-Scale Agile Development (see Appendix A). These studies may, to some extent, carry the bias of conference and workshop organizers and committee members. However, we did not address this type of bias as there is no effective way to determine whether such bias exists. Hence, we were not able to mitigate this kind of bias. Moreover, conferences and workshops, by definition, allow the publication of immature results that may distort the level of evidence of the selected studies.

6.4. Inaccuracy of data extraction

Data extraction bias may negatively affect the accuracy of data extraction results, affecting the classification results of the selected publications. Two researchers specified a list of extracted data items to mitigate this risk and reduce possible misunderstandings on the data items to be extracted. Two researchers also performed a pilot data extraction process before the formal data extraction. Further, two researchers conducted the main data extraction process in parallel and independently. Two researchers discussed and resolved conflicts arising from the data extraction results in several workshop sessions.

7. Conclusions and future work

Large-scale agile development is on the verge of becoming a mature research area, as many publications on large-scale agile development have appeared in scientific conferences and journals, leading to a growing body of knowledge. However, until now, no systematic mapping study has been published that systematically identifies, analyzes, and classifies the state of research. This mapping study aims to fill this gap and provide an overview of the research activities in large-scale agile development. Based on this objective, we selected 136 studies as a result of the systematic mapping process.

Our findings show that the adoption of agile methods has indeed inspired large companies around the world and in various sectors to apply these methods to larger projects and organizations. Our results suggest that this industrial interest has sparked significant academic interest in the topic of large-scale agile development, as a total of 136 articles were published in 46 publication venues by more than 200 authors worldwide between 2007 and 2019. In addition, our results show that research on large-scale agile development is mainly empirical and observational rather than solution-oriented, as most studies contribute in the form of lessons learned through case studies. Our results reveal that 10 research streams have emerged over time that focus on different aspects of large-scale agile development, e.g., agile architecture, scaling agile frameworks or team autonomy. Our findings show that the identified research streams raise many open research questions.

We hope that this mapping study will serve as a starting point for new research efforts that address previously neglected or emerging research topics and assist practitioners in the field of large-scale agile development. Based on our findings, we suggest that future research endeavors should pay greater attention to building a solid theoretical scaffold for the observed phenomena in large-scale agile development and should create rigorously developed frameworks, methods, and tools to meet practitioners' needs. Moreover, we recommend researchers to provide more conceptual clarity on the meaning of large-scale agile development, which has not received much attention but plays a crucial role in advancing the research field. We encourage researchers to use the compiled research questions to address the still open research gaps.

Appendix A. Distribution of selected studies by publication channels

#	Publication source	Type	Domain	No.	%
1	International Conference on Agile Software Development	Conference	Software engineering	21	15.44
2	International Workshop on Large-Scale Agile Development	Workshop	Software engineering	20	14.71
3	International Conference on Global Software Engineering	Conference	Software engineering	9	6.62
4	IEEE Software	Journal	Software engineering	6	4.41
5	Information and Software Technology	Journal	Software engineering	6	4.41
6	International Workshop on Autonomous Teams	Workshop	Software engineering	6	4.41
7	Empirical Software Engineering	Journal	Software engineering	5	3.68
8	Hawaii International Conference on System Sciences	Conference	Information systems	5	3.68
9	Americas Conference on Information Systems	Conference	Information systems	4	2.94
10	Journal of Systems and Software	Journal	Software engineering	4	2.94
11	Euromicro Conference on Software Engineering and Advanced Applications	Conference	Software engineering	3	2.21
12	International Conference on Enterprise Distributed Object Computing	Conference	Enterprise computing	3	2.21
13	International Symposium on Empirical Software Engineering and Measurement	Conference	Software engineering	3	2.21
14	Journal of Software: Evolution and Process	Journal	Software engineering	3	2.21
15	Project Management Journal	Journal	Project management	3	2.21
16	European Conference on Information Systems	Conference	Information systems	2	1.47
17	International Conference on Information Systems	Conference	Information systems	2	1.47
18	International Conference on Product-Focused Software Process Improvement	Conference	Software engineering	2	1.47
19	Software Process: Improvement and Practice	Journal	Software engineering	2	1.47
20	Bled eConference	Conference	Multidisciplinary	1	0.74
21	CIRP Design Conference	Conference	Multidisciplinary	1	0.74
22	European Conference on Pattern Languages of Programs	Conference	Multidisciplinary	1	0.74
23	Human Computer Interaction	Journal	Human computer interaction	1	0.74
24	IEEE Access	Journal	Multidisciplinary	1	0.74
25	IEEE Transactions on Software engineering	Journal	Software engineering	1	0.74
26	Information Systems Journal	Journal	Information systems	1	0.74
27	International Conference on Advanced Information Systems Engineering	Conference	Information systems	1	0.74
28	International Conference on Computing Communication and Automation	Conference	Multidisciplinary	1	0.74
29	International Conference on Information Systems Development	Conference	Information systems	1	0.74
30	International Conference on Pattern Languages of Programs	Conference	Multidisciplinary	1	0.74
31	International Conference on Perspectives in Business Informatics Research	Conference	Information systems	1	0.74
32	International Conference on Strategic Innovative Marketing	Conference	Marketing	1	0.74
33	International Journal of Information Systems and Project Management	Journal	Information systems	1	0.74
34	International Journal of Advanced Computer Science and Applications	Journal	Multidisciplinary	1	0.74
35	International Journal of Project Management	Journal	Project management	1	0.74
36	International Journal on Advanced Science, Engineering and Information Technology	Journal	Multidisciplinary	1	0.74
37	International Requirements Engineering Conference	Conference	Software engineering	1	0.74
38	International Research Workshop on IT Project Management	Workshop	IT project management	1	0.74
39	International Systems and Software Product Line Conference	Conference	Software engineering	1	0.74
40	International Working Conference on Requirements Engineering	Conference	Software engineering	1	0.74
41	International Workshop on Evidential Assessment of Software Technologies	Workshop	Software engineering	1	0.74
42	International Workshop on Requirements Engineering in Agile Development	Workshop	Software engineering	1	0.74
43	Malaysian Software Engineering Conference	Conference	Software engineering	1	0.74
44	Procedia Computer Science	Journal	Multidisciplinary	1	0.74
45	Software Quality Professional	Journal	Software engineering	1	0.74
46	Workshop on Agile Requirements Engineering	Workshop	Software engineering	1	0.74
	Total			136	100

Appendix B. Systematic map overview

Study	Publication type	Research type	Research approach	Contribution type	Research data type
[S1]	Journal	Evaluation research	Case study	Lessons learned	Primary study
[S2]	Journal	Evaluation research	Survey	Lessons learned	Primary study
[S3]	Journal	Evaluation research	Case study	Lessons learned	Primary study
[S4]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S5]	Workshop	Evaluation research	Case study	Lessons learned	Primary study
[S6]	Journal	Solution proposal	Design & creation	Framework/Method	Primary study
[S7]	Workshop	Philosophical papers	Case study	Model	Primary study
[S8]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S9]	Journal	Evaluation research	(Systematic) literature review	Lessons learned	Secondary study
[S10]	Conference	Evaluation research	(Systematic) literature review	Lessons learned	Secondary study
[S11]	Conference	Philosophical papers	Case study	Model	Primary study
[S12]	Workshop	Evaluation research	Case study	Lessons learned	Primary study
[S13]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S14]	Conference	Evaluation research	Grounded theory	Lessons learned	Primary study
[S15]	Conference	Solution proposal	Design & creation	Guideline	Primary study
[S16]	Workshop	Evaluation research	Mixed methods	Lessons learned	Primary study
[S17]	Workshop	Evaluation research	(Systematic) literature review	Lessons learned	Primary study
[S18]	Journal	Philosophical papers	Grounded theory	Theory	Primary study
[S19]	Journal	Solution proposal	Design & creation	Framework/Method	Primary study
[S20]	Workshop	Evaluation research	(Systematic) literature review	Lessons learned	Secondary study
[S21]	Conference	Philosophical papers	Case study	Model	Primary study
[S22]	Journal	Evaluation research	(Systematic) literature review	Lessons learned	Secondary study
[S23]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S24]	Journal	Solution proposal	Not stated	Framework/Method	Primary study
[S25]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S26]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S27]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S28]	Journal	Evaluation research	Case study	Lessons learned	Primary study
[S29]	Journal	Evaluation research	Grounded theory	Guideline	Primary study
[S30]	Conference	Philosophical papers	(Systematic) literature review	Lessons learned	Primary study
[S31]	Journal	Evaluation research	Case study	Lessons learned	Primary study
[S32]	Conference	Solution proposal	Case study	Model	Primary study
[S33]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S34]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S35]	Conference	Solution proposal	Design & creation	Framework/Method	Primary study
[S36]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S37]	Journal	Evaluation research	Case study	Lessons learned	Primary study
[S38]	Journal	Evaluation research	Case study	Lessons learned	Primary study
[S39]	Workshop	Evaluation research	Case study	Lessons learned	Primary study
[S40]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S41]	Conference	Solution proposal	Mixed methods	Framework/Method	Primary study
[S42]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S43]	Journal	Evaluation research	Case study	Lessons learned	Primary study
[S44]	Journal	Philosophical papers	Grounded theory	Model	Primary study
[S45]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S46]	Conference	Solution proposal	Design & creation	Framework/Method	Primary study
[S47]	Conference	Evaluation research	(Systematic) literature review	Lessons learned	Secondary study
[S48]	Conference	Evaluation research	Survey	Lessons learned	Primary study
[S49]	Journal	Evaluation research	Not stated	Guideline	Primary study
[S50]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S51]	Workshop	Evaluation research	Case study	Lessons learned	Primary study
[S52]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S53]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S54]	Workshop	Evaluation research	Case study	Lessons learned	Primary study
[S55]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S56]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S57]	Conference	Evaluation research	(Systematic) literature review	Lessons learned	Secondary study
[S58]	Conference	Evaluation research	Survey	Lessons learned	Primary study
[S59]	Workshop	Evaluation research	Not stated	Lessons learned	Primary study
[S60]	Journal	Evaluation research	Case study	Lessons learned	Primary study
[S61]	Workshop	Solution proposal	Grounded Theory	Model	Primary study
[S62]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S63]	Journal	Evaluation research	Case study	Lessons learned	Primary study
[S64]	Workshop	Evaluation research	Case study	Lessons learned	Primary study
[S65]	Journal	Evaluation research	(Systematic) literature review	Lessons learned	Secondary study
[S66]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S67]	Conference	Evaluation research	Mixed methods	Lessons learned	Primary study
[S68]	Journal	Evaluation research	Case study	Lessons learned	Primary study

Study	Publication type	Research type	Research approach	Contribution type	Research data type
[S69]	Conference	Evaluation research	Grounded theory	Lessons learned	Primary study
[S70]	Workshop	Solution proposal	Case study	Theory	Primary study
[S71]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S72]	Journal	Evaluation research	Mixed methods	Lessons learned	Primary study
[S73]	Conference	Philosophical papers	Mixed methods	Model	Primary study
[S74]	Workshop	Evaluation research	Case study	Lessons learned	Primary study
[S75]	Journal	Evaluation research	Survey	Lessons learned	Primary study
[S76]	Workshop	Evaluation research	Case study	Lessons learned	Primary study
[S77]	Workshop	Evaluation research	Not stated	Guideline	Primary study
[S78]	Journal	Evaluation research	Mixed methods	Lessons learned	Primary study
[S79]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S80]	Conference	Evaluation research	Grounded theory	Lessons learned	Primary study
[S81]	Journal	Evaluation research	Case study	Lessons learned	Primary study
[S82]	Journal	Evaluation research	Mixed methods	Lessons learned	Primary study
[S83]	Journal	Evaluation research	Case study	Lessons learned	Primary study
[S84]	Journal	Evaluation research	Mixed methods	Guideline	Primary study
[S85]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S86]	Workshop	Evaluation research	Case study	Lessons learned	Primary study
[S87]	Conference	Solution proposal	Case study	Framework/Method	Primary study
[S88]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S89]	Workshop	Evaluation research	(Systematic) literature review	Lessons learned	Secondary study
[S90]	Journal	Evaluation research	Grounded theory	Lessons learned	Primary study
[S91]	Conference	Evaluation research	Case study	Guideline	Primary study
[S92]	Journal	Evaluation research	Case study	Lessons learned	Primary study
[S93]	Conference	Philosophical papers	Theoretical	Lessons learned	Primary study
[S94]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S95]	Workshop	Evaluation research	Action research	Lessons learned	Primary study
[S96]	Journal	Evaluation research	Case study	Lessons learned	Primary study
[S97]	Journal	Evaluation research	Case study	Lessons learned	Primary study
[S98]	Workshop	Philosophical papers	Case study	Model	Primary study
[S99]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S100]	Journal	Solution proposal	Grounded Theory	Theory	Primary study
[S101]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S102]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S103]	Journal	Evaluation research	Case study	Lessons learned	Primary study
[S104]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S105]	Conference	Philosophical papers	Not stated	Model	Primary study
[S106]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S107]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S108]	Journal	Evaluation research	(Systematic) literature review	Lessons learned	Secondary study
[S109]	Workshop	Evaluation research	Case study	Lessons learned	Primary study
[S110]	Workshop	Solution proposal	Not stated	Guideline	Primary study
[S111]	Journal	Evaluation research	Mixed methods	Lessons learned	Primary study
[S112]	Journal	Evaluation research	Case study	Lessons learned	Primary study
[S113]	Workshop	Philosophical papers	Theoretical	Theory	Primary study
[S114]	Workshop	Evaluation research	Case study	Lessons learned	Primary study
[S115]	Workshop	Evaluation research	Case study	Lessons learned	Primary study
[S116]	Workshop	Evaluation research	Case study	Lessons learned	Primary study
[S117]	Workshop	Solution proposal	Design & creation	Guideline	Primary study
[S118]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S119]	Conference	Evaluation research	Survey	Lessons learned	Primary study
[S120]	Workshop	Evaluation research	Case study	Lessons learned	Primary study
[S121]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S122]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S123]	Workshop	Evaluation research	Survey	Lessons learned	Primary study
[S124]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S125]	Conference	Evaluation research	Mixed methods	Lessons learned	Primary study
[S126]	Conference	Evaluation research	Case study	lessons learned	Primary study
[S127]	Journal	Evaluation research	Theoretical	Guideline	Primary study
[S128]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S129]	Workshop	Evaluation research	Survey	Lessons learned	Primary study
[S130]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S131]	Conference	Evaluation research	Case study	Lessons learned	Primary study
[S132]	Conference	Evaluation research	(Systematic) literature review	Lessons learned	Secondary study
[S133]	Conference	Evaluation research	Not stated	Framework/Method	Primary study
[S134]	Conference	Evaluation research	(Systematic) literature review	Lessons learned	Secondary study
[S135]	Journal	Evaluation research	(Systematic) literature review	Lessons learned	Secondary study
[S136]	Conference	Evaluation research	(Systematic) literature review	Lessons learned	Secondary study

References

[1] Pekka Abrahamsson, Juhani Warsta, Mikko T Siponen, and Jussi Ronkainen. New directions on agile methods: A comparative analysis. In *Proceedings of the 25th International Conference on Software Engineering*, pages 244–254. IEEE, May 2003.

[2] Pekka Abrahamsson, Muhammad Ali Babar, and Philippe Kruchten. Agility and architecture: Can they coexist? *IEEE Software*, 27(2), 2010.

[3] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, and Juhani Warsta. Agile software development methods: Review and analysis. *arXiv preprint arXiv:1709.08439*, 2017.

[4] Muhammad Faisal Abrar, Muhammad Sohail Khan, Sikandar Ali, Umar Ali, Muhammad Faran Majeed, Amjad Ali, Bahrul Amin, and Nasir Rasheed. Motivators for large-scale agile adoption from management perspective: A systematic literature review. *IEEE Access*, 7:22660–22674, 2019.

[5] Mashal Alqudah and Rozilawati Razali. A review of scaling agile methods in large software development. *International Journal on Advanced Science, Engineering and Information Technology*, 6(6):828–837, 2016.

[6] Wasim Alsaqaf, Maya Daneva, and Roel Wieringa. Quality requirements in large-scale distributed agile projects – a systematic literature review. In *Requirements Engineering: Foundation for Software Quality*, pages 219–234, Cham, February 2017. Springer.

[7] Scott W. Ambler. Agile software development at scale. In *IFIP Central and East European Conference on Software Engineering Techniques*, pages 1–12. Springer, 2007.

[8] Scott W. Ambler and Mark Lines. *Disciplined agile delivery: A practitioner's guide to agile software delivery in the enterprise*. IBM Press, 2012.

[9] Stephen J Andriole. The death of big software. *Communications of the ACM*, 60(12):29–32, 2017.

[10] Muhammad Ali Babar. An exploratory study of architectural practices and challenges in using agile software development approaches. In *Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on*, pages 81–90. IEEE, 2009.

[11] Victor Basili, Gianluigi Caldiera, and Dieter Rombach. The goal question metric approach. *Encyclopedia of Software Engineering*, pages 528–532, 1994.

[12] Dinesh Batra. Research challenges and opportunities in conducting quantitative studies on large-scale agile methodology. *Journal of Database Management (JDM)*, 31(2):64–73, 2020.

[13] Kent Beck. *Extreme programming explained: embrace change*. addison-wesley professional, 2000.

[14] Kent Beck. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2000.

[15] Mike Beedle, Martine Devos, Yonat Sharon, Ken Schwaber, and Jeff Sutherland. Scrum: An extension pattern language for hyperproductive software development. *Pattern Languages of Program Design*, 4:637–651, 1999.

[16] Mike Beedle, James O. Coplien, Jeff Sutherland, Jens C. Østergaard, Ademar Aguiar, and Ken Schwaber. Essential scrum patterns. In *14th European Conference on Pattern Languages of Programs*, pages 1–17, Irsee, 2010. The Hillside Group.

[17] Vebjørn Berg, Jørgen Birkeland, Anh Nguyen-Duc, Ilias O. Pappas, and Letizia Jaccheri. Software startup engineering: A systematic mapping study. *Journal of Systems and Software*, 144:255–274, 2018.

[18] Hilary Berger and Paul Beynon-Davies. The utility of rapid application development in large-scale, complex projects. *Information Systems Journal*, 19(6):549–570, 2009.

[19] Elizabeth Bjarnason, Krzysztof Wnuk, and Björn Regnell. A case study on benefits and side-effects of agile practices in large-scale requirements engineering. In *Proceedings of the 1st Workshop on Agile Requirements Engineering (AREW)*, pages 1–5. Association for Computing Machinery (ACM), July 2011.

[20] Barry Boehm. Get ready for agile methods, with care. *Computer*, 35(1): 64–69, 2002.

[21] O. Pearl Brereton, Barbara A. Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, 80(4):571–583, 2007.

[22] David Budgen, Mark Turner, O. Pearl Brereton, and Barbara A. Kitchenham. Using mapping studies in software engineering. In *Psychology of Programming Interest Group*, volume 8, pages 195–204, 2008.

[23] The LeSS Company B.V. Large Scale Scrum. `https://less.works/case-studies`, 2022. [Online; accessed 18-APR-2022].

[24] Noel Carroll and Kieran Conboy. Applying normalization process theory to explain large-scale agile transformations. In *Proceedings of the 14th International Research Workshop on IT Project Management*, January 2019.

[25] Lianipng Chen, Muhammad Ali Babar, and He Zhang. Towards an evidence-based understanding of electronic data sources. In *Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 1–4. BCS Learning & Development Ltd, April 2010.

[26] Tsun Chow and Dac-Buu Cao. A survey study of critical success factors in agile software projects. *Journal of systems and software*, 81(6):961–971, 2008.

[27] James O. Coplien and Neil B. Harrison. *Organizational Patterns of Agile Software Development*. Addison-Wesley, Boston, 2004.

[28] Daniela S. Cruzes and Tore Dybå. Recommended steps for thematic synthesis in software engineering. In *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*, pages 275–284. Institute of Electrical and Electronics Engineers (IEEE), September 2011.

[29] Digital.ai. 12th State of Agile Survey. `https://stateofagile.com/#ufh-i-613553652-12th-annual-state-of-agile-report/7027494`, 2018. [Online; accessed 21-JUL-2021].

[30] Digital.ai. 13th State of Agile Survey. `https://stateofagile.com/#ufh-i-613553418-13th-annual-state-of-agile-report/7027494`, 2019. [Online; accessed 21-JUL-2021].

[31] Digital.ai. 14th State of Agile Survey. `https://stateofagile.com/#ufh-i-615706098-14th-annual-state-of-agile-report/7027494`, 2020. [Online; accessed 21-JUL-2021].

[32] 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.

[33] Torgeir Dingsøyr and Nils Brede Moe. Research challenges in large-scale agile software development. *ACM SIGSOFT Software Engineering Notes*, 38(5):38–39, 2013.

[34] Torgeir Dingsøyr, Sridhar Nerur, Venugopal Balijepally, and Nils Brede Moe. A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, 85(6):1213 – 1221, 2012. ISSN 0164-1212. Special Issue: Agile Development.

[35] Torgeir Dingsøyr, Tor Erlend Fægri, and Juha Itkonen. What is large in large-scale? a taxonomy of scale for agile software development. In *Proceedings of the 15th International Conference on Product-Focused Software Process Improvement (PROFES)*, pages 273–276. Springer, December 2014.

[36] Torgeir Dingsøyr, Nils Brede Moe, Tor Erlend Fægri, and Eva Amdahl Seim. Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation. *Empirical Software Engineering*, 23(1):490–520, 2018.

[37] Torgeir Dingsøyr, Davide Falessi, and Ken Power. Agile development at scale: the next frontier. *IEEE Software*, 36(2):30–38, 2019.

[38] Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9):833 – 859, 2008. ISSN 0950-5849.

[39] Tore Dybå and Torgeir Dingsøyr. What do we know about agile software development? *IEEE Software*, 26(5):6–9, 2009.

[40] Henry Edison, Xiaofeng Wang, and Kieran Conboy. Comparing methods for large-scale agile software development: A systematic literature review. *IEEE Transactions on Software Engineering*, 2021.

[41] Sallyann Freudenberg and Helen Sharp. The top 10 burning research questions from practitioners. *Ieee Software*, 27(5):8–9, 2010.

[42] Christoph Fuchs and Thomas Hess. Becoming agile in the digital transformation: The process of a large-scale agile transformation. In *Proceedings of the 39th International Conference on Information Systems (ICIS)*, December 2018.

[43] Tomas Gustavsson. Assigned roles for inter-team coordination in large-scale agile development: A literature review. In *Proceedings of the 5th International Workshop on Large-Scale Agile Development (XP)*, pages 1–5. Association for Computing Machinery (ACM), May 2017.

[44] Amani Mahdi Mohammed Hamed and Hisham Abushama. Popular agile approaches in software development: Review and analysis. In *Proceedings of the 2013 International Conference on Computing, Electrical and Electronic Engineering (ICCEEE)*, pages 160–166. Institute of Electrical and Electronics Engineers (IEEE), August 2013.

[45] Jo E. Hannay, Dag I. K. Sjoberg, and Tore Dybå. A systematic review of theory use in software engineering experiments. *IEEE Transactions on Software Engineering*, 33(2):87–107, 2007.

[46] Geir K. Hanssen, Darja Smite, and Nils Brede Moe. Signs of agile trends in global software engineering research: A tertiary study. In *2011 IEEE Sixth International Conference on Global Software Engineering Workshop*, pages 17–23, 2011.

[47] Ville Heikkilä, Daniela Damian, Casper Lassenius, and Maria Paasivaara. A mapping study on requirements engineering in agile software development. In *2015 41st Euromicro conference on software engineering and advanced applications*, pages 199–207. IEEE, 2015.

[48] Steffen Herbold, Aynur Amirfallah, Fabian Trautsch, and Jens Grabowski. A systematic mapping study of developer social network research. *Journal of Systems and Software*, 171:110802, 2021. ISSN 0164-1212.

[49] Jim Highsmith and Alistair Cockburn. Agile software development: the business of innovation. *Computer*, 34(9):120–127, 2001.

[50] Peter Hodgkins and Luke Hohmann. Agile program management: Lessons learned from the verisign managed security services team. In *Agile 2007 (AGILE 2007)*, pages 194–199. Institute of Electrical and Electronics Engineers (IEEE), August 2007.

[51] Scaled Agile Inc. Scaled Agile Framework. `http://www.scaledagileframework.com/case-studies/`, 2022. [Online; accessed 18-APR-2022].

[52] Magne Jorgensen and Martin Shepperd. A systematic review of software development cost estimation studies. *IEEE Transactions on software engineering*, 33(1):33–53, 2006.

[53] Martin Kalenda, Petr Hyna, and Bruno Rossi. Scaling agile in large organizations: Practices, challenges, and success factors. *Journal of Software: Evolution and Process*, 30(10):e1954, 2018.

[54] Petri Kettunen. Extending software project agility with new product development enterprise agility. *Software Process: Improvement and Practice*, 12(6):541–548, November 2007.

[55] Barbara A. Kitchenham and O. Pearl Brereton. A systematic review of systematic review process research in software engineering. *Information and Software Technology*, 55(12):2049–2075, 2013.

[56] Barbara A. Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. Technical report, Keele University and University of Durham, EBSE, 2007.

[57] Barbara A. Kitchenham, Emilia Mendes, and Guilherme H. Travassos. Cross versus within-company cost estimation studies: A systematic review. *IEEE Transactions on Software Engineering*, 33(5):316–329, 2007.

[58] Barbara A. Kitchenham, David Budgen, and O. Pearl Brereton. Using mapping studies as the basis for further research - a participant-observer case study. *Information and Software Technology*, 53(6):638–651, 2011.

[59] Jil Klünder, Regina Hebig, Paolo Tell, Marco Kuhrmann, Joyce Nakatumba-Nabende, Rogardt Heldal, Stephan Krusche, Masud Fazal-Baqaie, Michael Felderer, Marcela Fabiana Genero Bocco, et al. Catching up with method and process practice: An industry-informed baseline for researchers. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 255–264. IEEE, 2019.

[60] Harry Koehnemann and Mark Coats. Experiences applying agile practices to large systems. In *Agile Conference*, pages 295–300, 2009.

[61] Dina Koutsikouri, Sabine Madsen, and Nataliya Berbyuk Lindström. Agile transformation: How employees experience and cope with transformative change. In *International Conference on Agile Software Development*, pages 155–163. Springer, Cham, 2020.

[62] Daryl Kulak and Hong Li. *The journey to enterprise agility: Systems thinking and organizational legacy*. Springer, 2017.

[63] Craig Larman. *Practices for scaling lean & Agile development: large, multisite, and offshore product development with large-scale scrum*. Pearson Education India, 2010.

[64] Dean Leffingwell, Ryan Martens, and Mauricio Zamora. Principles of agile architecture. *Leffingwell, LLC. and Rally Software Development Corp*, 2008.

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

[66] Jeffrey A Livermore. Factors that significantly impact the implementation of an agile software development methodology. *Journal of Software*, 3(4): 31–36, 2008.

[67] Aniket Mahanti. Challenges in enterprise adoption of agile methods-a survey. *Journal of Computing and Information technology*, 14(3):197–206, 2006.

[68] Chuck Maples. Enterprise agile transformation: The two-year wall. In *Proceedings of the 2009 Agile Conference*, pages 90–95. IEEE, August 2009.

[69] Subhas Chandra Misra, Vinod Kumar, and Uma Kumar. Identifying some critical changes required in adopting agile practices in traditional software development projects. *International Journal of Quality & Reliability Management*, 2010.

[70] Nils Brede Moe and Torgeir Dingsøyr. Emerging research themes and updated research agenda for large-scale agile development: a summary of the 5th international workshop at xp2017. In *Proceedings of the XP2017 Scientific Workshops*, pages 1–4. Association for Computing Machinery (ACM), May 2017.

[71] Sridhar P Nerur, Abdul A Rasheed, and Vivek Natarajan. The intellectual structure of the strategic management field: An author co-citation analysis. *Strategic Management Journal*, 29(3):319–336, 2008.

[72] Mahmood Niazi, Sajjad Mahmood, Mohammad Alshayeb, Mohammed Rehan Riaz, Kanaan Faisal, Narciso Cerpa, Siffat Ullah Khan, and Ita Richardson. Challenges of project management in global software development: A client-vendor analysis. *Information and Software Technology*, 80:1–19, 2016.

[73] Robert L Nord, Ipek Ozkaya, and Philippe Kruchten. Agile in distress: Architecture to the rescue. In *International Conference on Agile Software Development*, pages 43–57. Springer, 2014.

[74] BJ Briony Oates. Evidence-based information systems: A decade later. In *Proceedings of the 19th European Conference on Information Systems (ECIS)*. Association for Information Systems, June 2011.

[75] Helena Holmström Olsson, Eoin Ó. Conchúir, Pär J. Ågerfalk, and Brian Fitzgerald. Global software development challenges: A case study on temporal, geographical and socio-cultural distance. In *2006 IEEE International Conference on Global Software Engineering (ICGSE'06)*, pages 3–11. IEEE, 2006.

[76] Wanda J. Orlikowski. Improvising organizational transformation over time: A situated change perspective. *Information Systems Research*, 7(1): 63–92, 1996.

[77] Necmettin Ozkan and Ayca Tarhan. A review of scaling approaches to agile software development models. *Software Quality Professional*, 21(4): 11–20, 2019.

[78] Maria Paasivaara, Sandra Durasiewicz, and Casper Lassenius. Using scrum in a globally distributed project: a case study. *Software Process: Improvement and Practice*, 13(6):527–544, 2008.

[79] Nicolò Paternoster, Carmine Giardino, Michael Unterkalmsteiner, Tony Gorschek, and Pekka Abrahamsson. Software development in startup companies: A systematic mapping study. *Information and Software Technology*, 56(10):1200–1218, 2014.

[80] Kai Petersen and Claes Wohlin. A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. *Journal of Systems and Software*, 82(9):1479–1490, 2009.

[81] Kai Petersen and Claes Wohlin. The effect of moving from a plan-driven to an incremental software development approach with agile practices. *Empirical Software Engineering*, 15(6):654–693, 2010.

[82] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 68–77, Swindon, UK, June 2008. BCS Learning & Development Ltd.

[83] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, 2015.

[84] Abheeshta Putta, Maria Paasivaara, and Casper Lassenius. Benefits and challenges of adopting the scaled agile framework (safe): Preliminary results from a multivocal literature review. In *Product-Focused Software Process Improvement*, pages 334–351, Cham, November 2018. Springer.

[85] Mark Rajpal. Lessons learned from a failed attempt at distributed ag-

ile. In *Proceedings of the 17th International Conferences XP 2016: Agile Processes in Software Engineering and Extreme Programming*, pages 235–243, Cham, May 2016. Springer.

[86] Abbas Moshref Razavi and Rodina Ahmad. Agile development in large and distributed environments: A systematic literature review on organizational, managerial and cultural aspects. In *Proceedings of the 2014 8th. Malaysian Software Engineering Conference (MySEC)*, pages 216–221. Institute of Electrical and Electronics Engineers (IEEE), September 2014.

[87] Donald J Reifer, Frank Maurer, and Hakan Erdogmus. Scaling agile methods. *IEEE software*, 20(4):12–14, 2003.

[88] Pilar Rodríguez, Jouni Markkula, Markku Oivo, and Kimmo Turula. Survey on agile and lean usage in finnish software industry. In *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 139–148, New York, NY, USA, September 2012. Association for Computing Machinery (ACM). ISBN 9781450310567.

[89] Pilar Rodríguez, Alireza Haghighatkhah, Lucy Ellen Lwakatare, Susanna Teppola, Tanja Suomalainen, Juho Eskeli, Teemu Karvonen, Pasi Kuvaja, June M. Verner, and Markku Oivo. Continuous deployment of software intensive products and services: A systematic mapping study. *Journal of Systems and Software*, 123:263–291, 2017.

[90] Knut H. Rolland, Brian Fitzgerald, Torgeir Dingsøyr, and Klaas-Jan Stol. Problematizing agile in the large: Alternative assumptions for large-scale agile development. In *Proceedings of the 37th International Conference on Information Systems (ICIS)*, December 2016.

[91] Dominik Rost, Balthasar Weitzel, Matthias Naab, Torsten Lenhart, and Hartmut Schmitt. Distilling best practices for agile development from architecture methodology. In *European Conference on Software Architecture*, pages 259–267. Springer, 2015.

[92] Hina Saeeda, Hannan Khalid, Mukhtar Ahmed, Abu Sameer, and Fahim Arif. Systematic literature review of agile scalability for large scale projects. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 6(9):63–75, 2015.

[93] Christoph Tobias Schmidt, Srinivasa Ganesha Venkatesha, and Juergen Heymann. Empirical insights into the perceived benefits of agile software engineering practices: A case study from sap. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 84–92. Association for Computing Machinery (ACM), May 2014.

[94] Ken Schwaber and Mike Beedle. *Agile software development with Scrum*, volume 1. Prentice Hall Upper Saddle River, 2002.

[95] Mohammad Shameem, Chiranjeev Kumar, Bibhas Chandra, and Arif Ali Khan. Systematic review of success factors for scaling agile methods in global software development environment: A client-vendor perspective. In *Proceedings of the 24th Asia-Pacific Software Engineering Conference Workshops (APSEC)*, pages 17–24. Institute of Electrical and Electronics Engineers (IEEE), December 2017.

[96] Mohammad Shameem, Bibhas Chandra, Rakesh Ranjan Kumar, and Chiranjeev Kumar. A systematic literature review to identify human related challenges in globally distributed agile software development: towards a hypothetical model for scaling agile methodologies. In *Proceedings of the 2018 4th International Conference on Computing Communication and Automation (ICCCA)*, pages 1–7. Institute of Electrical and Electronics Engineers (IEEE), December 2018.

[97] Mary Shaw. Writing good software engineering research papers. In *Proceedings of the 25th International Conference on Software Engineering, 2003.*, pages 726–736. Institute of Electrical and Electronics Engineers (IEEE), May 2003.

[98] Stavros Stavru. A critical examination of recent industrial surveys on agile method usage. *Journal of Systems and Software*, 94:87 – 97, 2014. ISSN 0164-1212.

[99] Ömer Uludağ, Martin Kleehaus, Xian Xu, and Florian Matthes. Investigating the role of architects in scaling agile frameworks. In *Proceedings of the 21st IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 123–132. Institute of Electrical and Electronics Engineers (IEEE), October 2017.

[100] Ömer Uludağ, Martin Kleehaus, Christoph Caprano, and Florian Matthes. Identifying and structuring challenges in large-scale agile development based on a structured literature review. In *Proceedings of the 22nd IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 191–197. Institute of Electrical and Electronics Engineers (IEEE), October 2018.

[101] Ömer Uludağ, Abheeshta Putta, Maria Paasivaara, and Florian Matthes. Evolution of the agile scaling frameworks. In *Proceedings of the 22nd International Conference on Agile Software Development*, Cham, June 2021. Springer.

[102] Michael Unterkalmsteiner, Tony Gorschek, A. K. M. Moinul Islam, Chow Kian Cheng, Rahadian Bayu Permadi, and Robert Feldt. Evaluation and measurement of software process improvement—a systematic literature review. *IEEE Transactions on Software Engineering*, 38(2):398–424, 2011.

[103] Roel Wieringa, Neil Maiden, Nancy Mead, and Colette Rolland. Requirements engineering paper classification and evaluation criteria: A proposal and a discussion. *Requirements engineering*, 11(1):102–107, 2006.

[104] Chen Yang, Peng Liang, and Paris Avgeriou. A systematic mapping study on the combination of software architecture and agile development. *Journal of Systems and Software*, 111:157–184, 2016.

[105] He Zhang, Muhammad Ali Babar, and Paolo Tell. Identifying relevant studies in software engineering. *Information and Software Technology*, 53 (6):625–637, 2011.

Selected studies

[S1] Kai Petersen and Claes Wohlin. A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. *Journal of Systems and Software*, 82(9):1479 – 1490, 2009.

[S2] Maarit Laanti, Outi Salo, and Pekka Abrahamsson. Agile methods rapidly replacing traditional methods at nokia: A survey of opinions on agile transformation. *Information and Software Technology*, 53(3): 276 – 290, 2011.

[S3] Maria Paasivaara and Casper Lassenius. Communities of practice in a large distributed agile software development organization – case ericsson. *Information and Software Technology*, 56(12):1556 – 1577, 2014.

[S4] Daniel Gerster, Christian Dremel, and Prashant Kelker. "agile meets non-agile": Implications of adopting agile practices at enterprises. In *Proceedings of the 24th Americas Conference on Information Systems (AMCIS)*, pages 836 – 845.

[S5] Elizabeth Bjarnason, Krzysztof Wnuk, and Björn Regnell. A case study on benefits and side-effects of agile practices in large-scale requirements engineering. In *Proceedings of the 1st Workshop on Agile Requirements Engineering (AREW)*, page 3. Association for Computing Machinery (ACM), July 2011.

[S6] Asif Qumer and Brian Henderson-Sellers. A framework to support the evaluation, adoption and improvement of agile methods in practice. *Journal of Systems and Software*, 81(11):1899–1919, 2008.

[S7] Ken Power. A model for understanding when scaling agile is appropriate in large organizations. In *Proceedings of the 2nd International Workshop on Large-Scale Agile Development (XP)*, pages 83–92. Springer, 2014.

[S8] Ville Heikkilä, Kristian Rautiainen, and Slinger Jansen. A revelatory case study on scaling agile release planning. In *Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA)*, pages 289–296. Institute of Electrical and Electronics Engineers (IEEE), September 2010.

[S9] Mashal Alqudah and Rozilawati Razali. A review of scaling agile methods in large software development. *International Journal on Advanced Science, Engineering and Information Technology*, 6(6):828–837, 2016.

[S10] Mohammad Shameem, Bibhas Chandra, Rakesh Ranjan Kumar, and Chiranjeev Kumar. A systematic literature review to identify human related challenges in globally distributed agile software development: towards a hypothetical model for scaling agile methodologies. In *Proceedings of the 4th International Conference on Computing Communication and Automation (ICCCA)*, pages 1–7. Institute of Electrical and Electronics Engineers (IEEE), December 2018.

[S11] Robin Duijs, Pascal Ravesteyn, and Marlies van Steenbergen. Adaptation of enterprise architecture efforts to an agile environment. In *Proceedings of the 31st Bled eConference*, pages 389–400, June 2018.

[S12] Abdallah Salameh and Julian M. Bass. Spotify tailoring for promoting effectiveness in cross-functional autonomous squads. In *Proceedings*

of the 2nd International Workshop on Autonomous Teams (XP), pages 20–28. Springer, May 2019.

[S13] Maria Paasivaara. Adopting safe to scale agile in a globally distributed organization. In *Proceedings of the 12th IEEE International Conference on Global Software Engineering (ICGSE)*, pages 36–40. Institute of Electrical and Electronics Engineers (IEEE), May 2017.

[S14] Julian M. Bass. Agile method tailoring in distributed enterprises: Product owner teams. In *Proceedings of the 8th IEEE International Conference on Global Software Engineering (ICGSE)*, pages 154–163. Institute of Electrical and Electronics Engineers (IEEE), August 2013.

[S15] Bettina Horlach, Ingrid Schirmer, and Paul Drews. Agile portfolio management: Design goal and principles. In *Proceedings of the 27th European Conference on Information Systems (ECIS)*. AIS Electronic Library (AISeL), June 2019.

[S16] Aivars Šāblis and Darja Šmite. Agile teams in large-scale distributed context: Isolated or connected? In *Proceedings of the Scientific Workshop of XP2016*, pages 1–5. Association for Computing Machinery (ACM), May 2016.

[S17] Sven Theobald, Anna Schmitt, and Philipp Diebold. Comparing scaling agile frameworks based on underlying practices. In *Proceedings of the 7th International Workshop on Large-Scale Agile (XP)*, pages 88–96. Springer, May 2019.

[S18] Julian M. Bass. Artefacts and agile method tailoring in large-scale offshore software development programmes. *Information and Software Technology*, 75:1–16, 2016.

[S19] Oktay Turetken, Igor Stojanov, and Jos JM Trienekens. Assessing the adoption level of scaled agile development: a maturity model for scaled agile framework. *Journal of Software: Evolution and Process*, 29(6): e1796, 2017.

[S20] Tomas Gustavsson. Assigned roles for inter-team coordination in large-scale agile development: A literature review. In *Proceedings of the 5th International Workshop on Large-Scale Agile Development (XP)*, pages 1–5. Association for Computing Machinery (ACM), May 2017.

[S21] Christoph Fuchs and Thomas Hess. Becoming agile in the digital transformation: The process of a large-scale agile transformation. In *Proceedings of the 39th International Conference on Information Systems (ICIS)*, December 2018.

[S22] 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.

[S23] Nils Brede Moe, Bjørn Dahl, Viktoria Stray, Lina Sund Karlsen, and Stine Schjødt-Osmo. Team autonomy in large-scale agile. In *Proceedings of the 52nd Hawaii International Conference on System Sciences (HICSS)*. Hawaii International Conference on System Sciences, January 2019.

[S24] Petri Kettunen and Maarit Laanti. Combining agile software projects and large-scale organizational agility. *Software Process: Improvement and Practice*, 13(2):183–193, 2008.

[S25] Antonio Martini, Lars Pareto, and Jan Bosch. Communication factors for speed and reuse in large-scale agile software development. In *Proceedings of the 17th international software product line conference (SPLC)*, pages 42–51. Association for Computing Machinery (ACM), August 2013.

[S26] Felix Evbota, Eric Knauss, and Anna Sandberg. Scaling up the planning game: Collaboration challenges in large-scale agile product development. In Helen Sharp and Tracy Hall, editors, *Proceedings of the 17th International Conference on Agile Software Development (XP)*, pages 28–38. Springer, May 2016.

[S27] Ville Heikkilä, Maria Paasivaara, Casper Lassenius, and Christian Engblom. Continuous release planning in a large-scale scrum development organization at ericsson. In *Proceedings of the 14th International Conference of Agile Software Development (XP)*, pages 195–209. Springer, June 2013.

[S28] Torgeir Dingsøyr, Nils Brede Moe, and Eva Amdahl Seim. Coordinating knowledge work in multi-team programs: Findings from a large-scale agile development program. *Project Management Journal*, 49(6): 64—77, 2018.

[S29] Saskia Bick, Kai Spohrer, Rashina Hoda, Alexander Scheerer, and Armin Heinzl. Coordination challenges in large-scale software development: A case study of planning misalignment in hybrid settings. *IEEE Trans. Software Eng.*, 44(10):932–950, 2018.

[S30] Alexander Scheerer, Tobias Hildenbrand, and Thomas Kude. Coordination in large-scale agile software development: A multiteam systems perspective. In *Proceedings of the 47th Hawaii International Conference on System Sciences (HICSS)*, pages 4780–4788. Institute of Electrical and Electronics Engineers (IEEE), January 2014.

[S31] Torgeir Dingsøyr, Knut Rolland, Nils Brede Moe, and Eva Amdahl Seim. Coordination in multi-team programmes: An investigation of the group mode in large-scale agile software development. *Procedia Computer Science*, 121:123–128, 2017.

[S32] Sina Katharina Weiss and Philipp Brune. Crossing the boundaries – agile methods in large-scale, plan-driven organizations: A case study from the financial services industry. In Eric Dubois and Klaus Pohl, editors, *Proceedings of the 29th International Conference Advanced Information Systems Engineering (CAiSE)*, pages 380–393. Springer, June 2017.

[S33] Helena Holmström Olsson, Jan Bosch, and Hiva Alahyari. Customer-specific teams for agile evolution of large-scale embedded systems. In *Proceedings of the 39th Euromicro Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA)*, pages 82–89, September 2013.

[S34] Maria Paasivaara, Sandra Durasiewicz, and Casper Lassenius. Distributed agile development: Using scrum in a large project. In *Proceedings of the 3rd IEEE International Conference on Global Software Engineering, (ICGSE)*, pages 87–95. Institute of Electrical and Electronics Engineers (IEEE), August 2008.

[S35] Ömer Uludağ, Nina-Mareike Harders, and Florian Matthes. Documenting recurring concerns and patterns in large-scale agile development. In *Proceedings of the 24th European Conference on Pattern Languages of Programs (EuroPLoP)*, July 2019.

[S36] Tomas Gustavsson. Dynamics of inter-team coordination routines in large-scale agile software development. In *Proceedings of the 27th European Conference on Information Systems (ECIS)*, June 2019.

[S37] Muhammad Usman, Ricardo Britto, Lars-Ola Damm, and Jürgen Börstler. Effort estimation in large-scale software development: An industrial case study. *Information and Software Technology*, 99:21–40, 2018.

[S38] Maria Paasivaara and Casper Lassenius. Empower your agile organization: Community-based decision making in large-scale agile development at ericsson. *IEEE Software*, 36(2):64–69, 2019.

[S39] Jan Henrik Gundelsby. Enabling autonomous teams in large-scale agile through architectural principles. In *Proceedings of the 1st International Workshop on Autonomous Agile Teams (XP Companion)*, page 17. Association for Computing Machinery (ACM), May 2018.

[S40] Ramesh Lal and Tony Clear. Enhancing product and service capability through scaling agility in a global software vendor environment. In *Proceedings of the 13th Conference on Global Software Engineering (ICGSE)*, pages 59–68. Association for Computing Machinery (ACM), May 2018.

[S41] Ömer Uludağ, Sascha Nägele, and Matheus Hauder. Establishing architecture guidelines in large-scale agile development through institutional pressures: A single-case study. In *Proceedings of the 25th Americas Conference on Information Systems (AMCIS)*, pages 551 – 560, August 2019.

[S42] Maria Paasivaara, Ville Heikkilä, and Casper Lassenius. Experiences in scaling the product owner role in large-scale globally distributed scrum. In *Proceedings of the 7th International Conference on Global Software Engineering (ICGSE)*, pages 174–178. Institute of Electrical and Electronics Engineers (IEEE), August 2012.

[S43] Torgeir Dingsøyr, Nils Brede Moe, Tor Erlend Fægri, and Eva Amdahl Seim. Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation. *Empirical Software Engineering*, 23(1):490–520, 2018.

[S44] Viviane Santos, Alfredo Goldman, and Cleidson RB De Souza. Fostering effective inter-team knowledge sharing in agile software development. *Empirical Software Engineering*, 20(4):1006–1051, 2015.

[S45] Daniel Gerster, Christian Dremel, Walter Brenner, and Prashant Kelker. How enterprises adopt agile structures: A multiple-case study. In *Proceedings of the 52nd Hawaii International Conference on System Sciences (HICSS)*, pages 1–10, January 2019.

[S46] Ömer Uludağ, Martin Kleehaus, Christoph Caprano, and Florian Matthes. Identifying and structuring challenges in large-scale agile

development based on a structured literature review. In *Proceedings of the 22nd IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 191–197. Institute of Electrical and Electronics Engineers (IEEE), October 2018.

[S47] Ömer Uludağ and Florian Matthes. Identifying and documenting recurring concerns and best practices of agile coaches and scrum masters in large-scale agile development. In *Proceedings of the 26th International Conference on Pattern Languages of Programs (PLoP)*, volume 26, pages 191–197. Hillside Group, 2019.

[S48] Tomas Gustavsson. Impacts on team performance in large-scale agile software development. In *Proceedings of the 2018 Joint of the 17th Business Informatics Research Short Papers, Workshops and Doctoral Consortium (BIC)*, pages 421–431. CEUR-WS, September 2018.

[S49] Kieran Conboy and Noel Carroll. Implementing large-scale agile frameworks: Challenges and recommendations. *IEEE Software*, 36 (2):44–50, 2019.

[S50] Maria Paasivaara, Casper Lassenius, Ville Heikkilä, Kim-Karol Dikert, and Christian Engblom. Integrating global sites into the lean and agile transformation at ericsson. In *Proceedings of the 8th IEEE International Conference on Global Software Engineering, (ICGSE)*, pages 134–143. Institute of Electrical and Electronics Engineers (IEEE), August 2013.

[S51] Saskia Bick, Alexander Scheerer, and Kai Spohrer. Inter-team coordination in large agile software development settings: Five ways of practicing agile at scale. In *Proceedings of the Scientific Workshops of XP2016*, page 4. Association for Computing Machinery (ACM), May 2016.

[S52] Finn Olav Bjørnson, Julia Wijnmaalen, Christoph Johann Stettina, and Torgeir Dingsøyr. Inter-team coordination in large-scale agile development: A case study of three enabling mechanisms. In *Proceedings of the 19th International Conference on Agile Software Development (XP)*, pages 216–231. Springer, May 2018.

[S53] Maria Paasivaara, Casper Lassenius, and Ville T Heikkilä. Inter-team coordination in large-scale globally distributed scrum: Do scrum-of-scrums really work? In *Proceedings of the 6th ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM)*, pages 235–238. Association for Computing Machinery (ACM), September 2012.

[S54] Helga Nyrud and Viktoria Stray. Inter-team coordination mechanisms in large-scale agile. In *Proceedings of the Scientific Workshops of XP2017*, pages 1–6. Association for Computing Machinery (ACM), May 2017.

[S55] Ömer Uludağ, Martin Kleehaus, Niklas Dreymann, Christian Kabelin, and Florian Matthes. Investigating the adoption and application of large-scale scrum at a german automobile manufacturer. In *Proceedings of the 14th International Conference on Global Software Engineering (ICGSE)*, pages 22–29. Institute of Electrical and Electronics Engineers (IEEE), May 2019.

[S56] Ömer Uludağ, Henderik A Proper, and Florian Matthes. Investigating the establishment of architecture principles for supporting large-scale agile transformations. In *Proceedings of the 23rd IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 41–50. Institute of Electrical and Electronics Engineers (IEEE), October 2019.

[S57] Ömer Uludağ, Martin Kleehaus, Xian Xu, and Florian Matthes. Investigating the role of architects in scaling agile frameworks. In *Proceedings of the 21st IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 123–132. Institute of Electrical and Electronics Engineers (IEEE), October 2017.

[S58] Yngve Lindsjørn, Gunnar R. Bergersen, Torgeir Dingsøyr, and Dag I. K. Sjøberg. Teamwork quality and team performance: Exploring differences between small and large agile projects. In *Proceedings of the 19th International Conference on Agile Software Development (XP)*, pages 267–274. Springer, 2018.

[S59] Finn Olav Bjørnson and Kathrine Vestues. Knowledge sharing and process improvement in large-scale agile development. In *Proceedings of the Scientific Workshops of XP2016*, page 7. Association for Computing Machinery (ACM), May 2016.

[S60] Maria Paasivaara, Benjamin Behm, Casper Lassenius, and Minna Hallikainen. Large-scale agile transformation at ericsson: a case study. *Empirical Software Engineering*, 23(5):2550–2596, 2018.

[S61] Julian M. Bass. Large-scale offshore agile tailoring: Exploring product and service organisations. In *Proceedings of the Scientific Workshops of XP2016*, page 8. Association for Computing Machinery (ACM), May 2016.

[S62] Torgeir Dingsøyr, Marius Mikalsen, Anniken Solem, and Kathrine Vestues. Learning in the large - an exploratory study of retrospectives in large-scale agile development. In *Proceedings of the 19th International Conference on Agile Software Development (XP)*, pages 191–198. Springer, May 2018.

[S63] Ville T. Heikkilä, Maria Paasivaara, Casper Lassenius, Daniela E. Damian, and Christian Engblom. Managing the requirements flow from strategy to release in large-scale agile development: a case study at ericsson. *Empirical Software Engineering*, 22(6):2892–2936, 2017.

[S64] Antonio Martini, Viktoria Stray, and Nils Brede Moe. Technical-, social- and process debt in large-scale agile: An exploratory case-study. In Rashina Hoda, editor, *Proceedings of the 7th International Workshop on Large-Scale Agile (XP)*, pages 112–119, Cham, May 2019. Springer.

[S65] Muhammad Faisal Abrar, Muhammad Sohail Khan, Sikandar Ali, Umar Ali, Muhammad Faran Majeed, Amjad Ali, Bahrul Amin, and Nasir Rasheed. Motivators for large-scale agile adoption from management perspective: A systematic literature review. *IEEE Access*, 7: 22660–22674, 2019.

[S66] Georgios Papadopoulos. Moving from traditional to agile software development methodologies also on large, distributed projects. *Procedia-Social and Behavioral Sciences*, 175:455–463, 2015.

[S67] Nils Brede Moe, Darja Šmite, Aivars Šāblis, Anne-Lie Börjesson, and Pia Andréasson. Networking in a large-scale distributed agile project. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–8. Association for Computing Machinery (ACM), September 2014.

[S68] Ville Heikkilä, Maria Paasivaara, Kristian Rautiainen, Casper Lassenius, Towo Toivola, and Janne Järvinen. Operational release planning in large-scale scrum with multiple stakeholders - A longitudinal case study at f-secure corporation. *Information and Software Technology*, 57:116–140, 2015.

[S69] Deepika Badampudi, Samuel A Fricker, and Ana M Moreno. Perspectives on productivity and delays in large-scale agile projects. In *Proceedings of the 14th International Conference on Agile Software Development (XP)*, pages 180–194. Springer, June 2013.

[S70] Viktoria Stray. Planned and unplanned meetings in large-scale projects. In *Proceedings of the 6th International Workshop on Large-scale Agile Development (XP Companion)*, pages 1–5. Association for Computing Machinery (ACM), May 2018.

[S71] Tomas Gustavsson. Practices for vertical and horizontal coordination in the scaled agile framework. In *Proceedings of the 27th International Conference on Information Systems Development (ISD)*, August 2018.

[S72] Mohammad Shameem, Rakesh Ranjan Kumar, Chiranjeev Kumar, Bibhas Chandra, and Arif Ali Khan. Prioritizing challenges of agile process in distributed software development environment using analytic hierarchy process. *Journal of Software: Evolution and Process*, 30(11), 2018.

[S73] Knut H. Rolland, Brian Fitzgerald, Torgeir Dingsøyr, and Klaas-Jan Stol. Problematizing agile in the large: Alternative assumptions for large-scale agile development. In *Proceedings of the 37th International Conference on Information Systems (ICIS)*, December 2016.

[S74] Tor Erlend Fægri and Nils Brede Moe. Re-conceptualizing requirements engineering: findings from a large-scale, agile project. In *Proceedings of the 1st International Workshop on Requirements Engineering in Agile Development (READ)*, page 4. Association for Computing Machinery (ACM), May 2015.

[S75] Magne Jørgensen. Relationships between project size, agile practices, and successful software development: Results and analysis. *IEEE Software*, 36(2):39–43, 2019.

[S76] Knut H. Rolland. Scaling across knowledge boundaries: A case study of A large-scale agile software development project. In *Proceedings of the Scientific Workshops of XP2016*, page 5. Association for Computing Machinery (ACM), May 2016.

[S77] Philipp Diebold, Anna Schmitt, and Sven Theobald. Scaling agile: how to select the most appropriate framework. In *Proceedings of the 6th International Workshop on Large-scale Agile Development (XP Compan-*

ion), pages 1–4. Association for Computing Machinery (ACM), May 2018.

[S78] Martin Kalenda, Petr Hyna, and Bruno Rossi. Scaling agile in large organizations: Practices, challenges, and success factors. *Journal of Software: Evolution and Process*, 30(10):e1954, 2018.

[S79] Maria Paasivaara and Casper Lassenius. Scaling scrum in a large globally distributed organization: A case study. In *Proceedings of the 11th IEEE International Conference on Global Software Engineering (ICGSE)*, pages 74–83. Institute of Electrical and Electronics Engineers (IEEE), August 2016.

[S80] Julian M. Bass. Scrum master activities: Process tailoring in large enterprise projects. In *Proceedings of the 9th IEEE International Conference on Global Software Engineering (ICGSE)*, pages 6–15. Institute of Electrical and Electronics Engineers (IEEE), August 2014.

[S81] Ricardo Britto, Darja Šmite, and Lars-Ola Damm. Software architects in large-scale distributed projects: An ericsson case study. *IEEE Software*, 33(6):48–55, 2016.

[S82] Helena Tendedez, Maria Angela MAF Ferrario, and Jon Whittle. Software development and cscw: Standardization and flexibility in large-scale agile development. pages 1–23. Association for Computing Machinery (ACM), November 2018.

[S83] Darja Šmite, Nils Brede Moe, Aivars Šāblis, and Claes Wohlin. Software teams and their knowledge networks in large-scale software development. *Information and Software Technology*, 86:71–86, 2017.

[S84] Darja Šmite, Nils Brede Moe, Georgiana Levinta, and Marcin Floryan. Spotify guilds: How to succeed with knowledge sharing in large-scale agile organizations. *IEEE Software*, 36(2):51–57, 2019.

[S85] Marthe Berntzen, Nils Brede Moe, and Viktoria Stray. The product owner in large-scale agile: An empirical study through the lens of relational coordination theory. In *Proceedings of the 20th International Conference on Agile Software Development (XP)*, pages 121–136. Springer, May 2019.

[S86] Maria Paasivaara, Outi Väättänen, Minna Hallikainen, and Casper Lassenius. Supporting a large-scale lean and agile transformation by defining common values. In *Proceedings of the XP 2014 International Workshops (XP)*, pages 73–82. Springer, May 2014.

[S87] Ömer Uludağ, Matheus Hauder, Martin Kleehaus, Christina Schimpfle, and Florian Matthes. Supporting large-scale agile development with domain-driven design. In *Proceedings of the 19th International Conference on Agile Software Development (XP)*, pages 232–247. Springer, May 2018.

[S88] Kristian Rautiainen, Joachim von Schantz, and Jarno Vähäniitty. Supporting scaling agile with portfolio management: Case paf.com. In *Proceedings of the 44th Hawaii International Conference on System Sciences (HICSS)*, pages 1–10. Institute of Electrical and Electronics Engineers (IEEE), January 2011.

[S89] Mohammad Shameem, Chiranjeev Kumar, Bibhas Chandra, and Arif Ali Khan. Systematic review of success factors for scaling agile methods in global software development environment: A client-vendor perspective. In *Proceedings of the 24th Asia-Pacific Software Engineering Conference Workshops (APSEC)*, pages 17–24. Institute of Electrical and Electronics Engineers (IEEE), December 2017.

[S90] Julian M. Bass and Andy Haxby. Tailoring product ownership in large-scale agile projects: Managing scale, distance, and governance. *IEEE Software*, 36(2):58–63, 2019.

[S91] Nelson Sekitoleko, Felix Evbota, Eric Knauss, Anna Sandberg, Michel Chaudron, and Helena Holmström Olsson. Technical dependency challenges in large-scale agile software development. In *Proceedings of the 15th International Conference on Agile Software Development (XP)*, pages 46–61. Springer, May 2014.

[S92] Kai Petersen and Claes Wohlin. The effect of moving from a plan-driven to an incremental software development approach with agile practices: An industrial case study. *Empirical Software Engineering*, 15(6):654–693, 12 2010.

[S93] Alexander Scheerer, Saskia Bick, Tobias Hildenbrand, and Armin Heinzl. The effects of team backlog dependencies on agile multiteam systems: A graph theoretical approach. In *Proceedings of the 48th Hawaii International Conference on System Sciences (HICSS)*, pages 5124–5132. Institute of Electrical and Electronics Engineers (IEEE), January 2015.

[S94] Lina Lagerberg, Tor Skude, Pär Emanuelsson, Kristian Sandahl, and Daniel Stahl. The impact of agile principles and practices on large-scale software development projects: A multiple-case study of two projects at ericsson. In *Proceedings of the 7th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 348–356. Institute of Electrical and Electronics Engineers (IEEE), October 2013.

[S95] Jan Pries Heje and Malene M Krohn. The safe way to the agile organization. In *Proceedings of the 5th International Workshop on Large-Scale Agile Development (XP)*, page 18, May 2017.

[S96] Hilary Berger and Paul Beynon-Davies. The utility of rapid application development in large-scale, complex projects. *Information Systems Journal*, 19(6):549–570, 2009.

[S97] Nils Brede Moe, Torgeir Dingsøyr, and Knut Rolland. To schedule or not to schedule? an investigation of meetings as an inter-team coordination mechanism in large-scale agile software development. *IJISPM - International Journal of Information Systems and Project Management*, 6(3):45–59, 2018.

[S98] Helena Holmström Olsson and Jan Bosch. Towards continuous validation of customer value. In *Proceedings of the 3rd International Workshop on Large-Scale Agile Development (XP)*, page 3. Association for Computing Machinery (ACM), May 2015.

[S99] Maria Paasivaara, Benjamin Behm, Casper Lassenius, and Minna Hallikainen. Towards rapid releases in large-scale xaas development at ericsson: A case study. In *Proceedings of the 9th IEEE International Conference on Global Software Engineering (ICGSE)*, pages 16–25. Institute of Electrical and Electronics Engineers (IEEE), August 2014.

[S100] Miloš Jovanović, Antònia Mas, Antoni-Lluís Mesquida, and Bojan Lalić. Transition of organizational roles in agile transformation process: A grounded theory approach. *Journal of Systems and Software*, 133:174–194, 2017.

[S101] Wasim Alsaqaf, Maya Daneva, and Roel Wieringa. Understanding challenging situations in agile quality requirements engineering and their solution strategies: Insights from a case study. In *Proceedings of the 26th IEEE International Requirements Engineering Conference (RE)*, pages 274–285. Institute of Electrical and Electronics Engineers (IEEE), August 2018.

[S102] B Veeresh Thummadi, Vishal D. Khapre, and Rosalie J. Ocker. Unpacking agile enterprise architecture innovation work practices: A qualitative case study of a railroad company. In *Proceedings of the 23rd Americas Conference on Information Systems (AMCIS)*, pages 3782 – 3791, August 2017.

[S103] Maria Paasivaara, Sandra Durasiewicz, and Casper Lassenius. Using scrum in a globally distributed project: a case study. *Software Process: Improvement and Practice*, 13(6):527–544, 2008.

[S104] Ömer Uludağ, Martin Kleehaus, Soner Erçelik, and Florian Matthes. Using social network analysis to investigate the collaboration between architects and agile teams: A case study of a large-scale agile development program in a german consumer electronics company. In *Proceedings of the 20th International Conference on Agile Software Development (XP)*, pages 137–153, May 2019.

[S105] Torgeir Dingsøyr, Tor Erlend Fægri, and Juha Itkonen. What is large in large-scale? a taxonomy of scale for agile software development. In *Proceedings of the 15th International Conference on Product-Focused Software Process Improvement (PROFES)*, pages 273–276. Springer, December 2014.

[S106] Ömer Uludağ, Martin Kleehaus, Niklas Reiter, and Florian Matthes. What to expect from enterprise architects in large-scale agile development? A multiple-case study. In *Proceedings of the 25th Americas Conference on Information Systems (AMCIS)*, pages 2683 – 2692, August 2019.

[S107] Antonio Martini, Lars Pareto, and Jan Bosch. Towards introducing agile architecting in large companies: The caffea framework. In *Proceedings of the 16th International Conference on Agile Software Development (XP)*, pages 218–223. Springer, May 2015.

[S108] Necmettin Ozkan and Ayca Tarhan. A review of scaling approaches to agile software development models. *Software Quality Professional*, 21 (4):11–20, 2019.

[S109] Marius Mikalsen, Magne Næsje, Erik André Reime, and Anniken Solem. Agile autonomous teams in complex organizations. In *Proceedings of the 2nd International Workshop on Autonomous Teams (XP)*, pages 55–63. Springer, May 2019.

[S110] Robert L. Nord, Ipek Ozkaya, and Philippe Kruchten. Agile in distress: Architecture to the rescue. In *Proceedings of the XP 2014 International Workshops*, pages 43–57. Springer, May 2014.

[S111] Brian Hobbs and Yvan Petit. Agile methods on large projects in large organizations. *Project Management Journal*, 48(3):3–19, 2017.

[S112] Christoph Johann Stettina and Jeannette Hörz. Agile portfolio management: An empirical perspective on the practice in use. *International Journal of Project Management*, 33(1):140 – 152, 2015.

[S113] Noel Carroll and Kieran Conboy. Applying normalization process theory to explain large-scale agile transformations. In *Proceedings of the 14th International Research Workshop on IT Project Management (IRWITPM)*, January 2019.

[S114] Yngve Lindsjørn and Roza Moustafa. Challenges with lack of trust in agile projects with autonomous teams and fixed-priced contracts. In *Proceedings of the 1st International Workshop on Autonomous Agile Teams (XP Companion)*, pages 1–5, May 2018.

[S115] Tomas Gustavsson. Changes over time in a planned inter-team coordination routine. In *Proceedings of the 7th International Workshop on Large-Scale Agile (XP)*, pages 105–111. Springer, May 2019.

[S116] Tomas Gustavsson. Voices from the teams - impacts on autonomy in large-scale agile software development settings.

[S117] Jaana Nyfjord, Sameer Bathallath, and Harald Kjellin. Conventions for coordinating large agile projects. In *Proceedings of the XP 2014 International Workshops (XP)*, pages 58–72. Springer, May 2014.

[S118] Darja Šmite, Nils Brede Moe, Jonas Wigander, and Hendrik Esser. Corporate-level communities at ericsson: Parallel organizational structure for fostering alignment for autonomy. In *Proceedings of the 20th International Conference on Agile Software Development (XP)*, pages 173–188. Springer, May 2019.

[S119] Magne Jørgensen. Do agile methods work for large software projects? In Juan Garbajosa, Xiaofeng Wang, and Ademar Aguiar, editors, *Proceedings of the 19th International Conference on Agile Software Development (XP)*, pages 179–190. Springer, May 2018.

[S120] Yvan Petit and Carl Marnewick. Earn your wings: A novel approach to deployment governance. In Rashina Hoda, editor, *Proceedings of the 2nd International Workshop on Autonomous Teams (XP)*, pages 64–71. Springer, May 2019.

[S121] Leonor Barroca, Helen Sharp, Torgeir Dingsøyr, Peggy Gregory, Katie Taylor, and Raid AlQaisi. Enterprise agility: A balancing act - a local government case study. In *Proceedings of the 20th International Conference on Agile Softeware Development (XP)*, pages 207–223. Springer, May 2019.

[S122] Teemu Karvonen, Helen Sharp, and Leonor Barroca. Enterprise agility: Why is transformation so hard? In *Proceedings of the 19th International Conference on Agile Softeware Development (XP)*, pages 131–145. Springer, May 2018.

[S123] Petri Kettunen, Maarit Laanti, Fabian Fagerholm, Tommi Mikkonen, and Tomi Männistö. Finnish enterprise agile transformations: A survey study. In *Proceedings of the 7th International Workshop on Large-Scale Agile (XP)*, pages 97–104. Springer, May 2019.

[S124] Abheeshta Putta, Maria Paasivaara, and Casper Lassenius. How are agile release trains formed in practice? a case study in a large financial corporation. In *Proceedings of the 20th International Conference on Agile Software Development (XP)*, pages 154–170. Springer, May 2019.

[S125] Sven Theobald and Philipp Diebold. Interface problems of agile in a non-agile environment. In *Proceedings of the 19th International Conference on Agile Software Development (XP)*, pages 123–130. Springer, May 2018.

[S126] Bjørnar Tessem and Frank Maurer. Job satisfaction and motivation in a large agile team. In *Proceedings of the 8th International Conference on Agile Software Development (XP)*, pages 54–61. Springer, June 2007.

[S127] Roger Sweetman and Kieran Conboy. Portfolios of agile projects: A complex adaptive systems' agent perspective. *Project Management Journal*, 49(6):18–38, 2018.

[S128] Christoph Johann Stettina and Lennard Schoemaker. Reporting in agile portfolio management: Routines, metrics and artefacts to maintain an effective oversight. In *Proceedings of the 19th International Conference on Agile Software Development (XP)*, pages 199–215. Springer, May 2018.

[S129] Maarit Laanti and Petri Kettunen. Safe adoptions in finland: A survey research. In *Proceedings of the 7th International Workshop on Large-Scale Agile (XP)*, pages 81–87. Springer, May 2019.

[S130] Iris Figalist, Christoph Elsner, Jan Bosch, and Helena Holmström Olsson. Scaling agile beyond organizational boundaries: Coordination challenges in software ecosystems. In *Proceedings of the 20th International Conference on Agile Software Development (XP)*, pages 189–206. Springer, May 2019.

[S131] Nirnaya Tripathi, Pilar Rodríguez, Muhammad Ovais Ahmad, and Markku Oivo. Scaling kanban for software development in a multi-site organization: Challenges and potential solutions. In *Proceedings of the 16th International Conference on Agile Software Development (XP)*, pages 178–190. Springer, May 2015.

[S132] Abheeshta Putta, Maria Paasivaara, and Casper Lassenius. Benefits and challenges of adopting the scaled agile framework (safe): Preliminary results from a multivocal literature review. In *Product-Focused Software Process Improvement*, pages 334–351, Cham, November 2018. Springer.

[S133] Günther Schuh, Eric Rebentisch, Christian Dölle, Christian Mattern, Georgiy Volevach, and Alexander Menges. Defining scaling strategies for the improvement of agility performance in product development projects. *Procedia CIRP*, 70:29–34, May 2018. 28th CIRP Design Conference 2018, 23-25 May 2018, Nantes, France.

[S134] Wasim Alsaqaf, Maya Daneva, and Roel Wieringa. Quality requirements in large-scale distributed agile projects – a systematic literature review. In *Requirements Engineering: Foundation for Software Quality*, pages 219–234, Cham, February 2017. Springer.

[S135] Hina Saeeda, Hannan Khalid, M. Ahmed, Abu Sameer, and Fahim Arif. Systematic literature review of agile scalability for large scale projects. *International Journal of Advanced Computer Science and Applications*, 6(9):63–75, 2015.

[S136] Abbas Moshref Ravazi and Rodina Ahmad. Agile development in large and distributed environments: A systematic literature review on organizational, managerial and cultural aspects. In *Proceedings of the 8th Malaysian Software Engineering Conference (MySEC)*, pages 216–221. Institute of Electrical and Electronics Engineers (IEEE), September 2014.

Selected workshop summaries

[W1] Torgeir Dingsøyr and Nils Brede Moe. Research challenges in large-scale agile software development. *ACM SIGSOFT Software Engineering Notes*, 38(5):38–39, 2013.

[W2] Torgeir Dingsøyr and Nils Brede Moe. Towards principles of large-scale agile development - A summary of the workshop at XP2014 and a revised research agenda. In *Proceedings of the XP 2014 International Workshops (XP)*, pages 1–8. Springer, May 2014.

[W3] Nils Brede Moe, Helena Holmström Olsson, and Torgeir Dingsøyr. Trends in large-scale agile development: A summary of the 4th workshop at XP2016. In *Proceedings of the Scientific Workshop of XP2016*, page 1. Association for Computing Machinery (ACM), May 2016.

[W4] Nils Brede oe and Torgeir Dingsøyr. Emerging research themes and updated research agenda for large-scale agile development: a summary of the 5th international workshop at XP2017. In *Proceedings of the 5th International Workshop on Large-Scale Agile Development (XP)*, pages 1–4. Association for Computing Machinery (ACM), May 2017.

[W5] Torgeir Dingsøyr, Nils Brede Moe, and Helena Holmström Olsson. Towards an understanding of scaling frameworks and business agility: a summary of the 6th international workshop at XP2018. In *Proceedings of the 6th International Workshop on Large-scale Agile Development (XP Companion)*, pages 1–4. Association for Computing Machinery (ACM), May 2018.

[W6] Julian M. Bass. Future trends in agile at scale: A summary of the 7th international workshop on large-scale agile development. In *Proceedings of the 7th International Workshop on Large-Scale Agile (XP)*, pages 75–80. Springer, May 2019.

[W7] Leonor Barroca, Torgeir Dingsøyr, and Marius Mikalsen. Agile transformation: A summary and research agenda from the first international workshop. In *Proceedings of the 1st International Workshop on Agile Transformation (XP)*, pages 3–9. Springer, May 2019.

[W8] Viktoria Stray, Nils Brede Moe, and Rashina Hoda. Autonomous agile teams: challenges and future directions for research. In *Proceedings of the 1st International Workshop on Autonomous Agile Teams (XP Companion)*, pages 1–5. Association for Computing Machinery (ACM), May 2018.

[W9] Nils Brede Moe, Viktoria Stray, and Rashina Hoda. Trends and updated research agenda for autonomous agile teams: A summary of the second international workshop at XP2019. In *Proceedings of the 2nd International Workshop on Autonomous Teams (XP)*, pages 13–19. Springer, May 2019.

Investigating the Role of Architects in Scaling Agile Frameworks

Ömer Uludağ, Martin Kleehaus, Xian Xu, Florian Matthes
Chair for Informatics 19
Technische Universität München (TUM)
D-85748, Garching
{*oemer.uludag,martin.kleehaus,xian.xu,matthes*}*@tum.de*

***Abstract*—This study describes the roles of architects in scaling agile frameworks with the help of a structured literature review. We aim to provide a primary analysis of 20 identified scaling agile frameworks. Subsequently, we thoroughly describe three popular scaling agile frameworks: Scaled Agile Framework, Large Scale Scrum, and Disciplined Agile 2.0. After specifying the main concepts of scaling agile frameworks, we characterize roles of enterprise, software, solution, and information architects, as identified in four scaling agile frameworks. Finally, we provide a discussion of generalizable findings on the role of architects in scaling agile frameworks.**

***Keywords*-scaling agile frameworks; agile software development; application architecture;**

I. Introduction

Enterprises struggle to deal with unpredictable competitive environments due to rapidly changing customer demands, regulatory changes, and technological advancements that can lead to the enterprise's success [1], [2]. Thus, the ability to detect relevant changes and respond in a timely and effective manner becomes an important determinant of the enterprise's survival [3]. Software development projects in such environments face changes either directly or indirectly. In order to face these challenges, the agile movement emerged in the 1990s, leading to the development of agile manifesto and many agile software development methods, including extreme programming (XP), kanban, and scrum [4]. With these agile methods, small, co-located, self-organizing teams work closely with the business customer on a single-project context, maximizing the customer value and quality of the delivered software product through rapid iterations and frequent feedback loops [4].

Since the initial application of these methods are tailored for small teams, large enterprises are interested in extending agile methods to include larger teams and inter-team coordination and communication [5]. Various scaling agile frameworks, e.g., the Scaled Agile Framework (SAFe), Disciplined Agile 2.0 (DA 2.0), and Large-Scale Scrum (LeSS), were proposed to resolve issues associated with team size, customer involvement, and project constraints [6].

Agile methods imply that the architecture should evolve incrementally and constantly be tested with known requirements rather than being imposed by some direct structuring force (emergent architecture design) [7]. The practice of emergent architecture design is effective at the team level but insufficient when developing large systems. It causes excessive redesign efforts, architectural divergence, and functional redundancy increasing the complexity of application architectures [7], [8]. Therefore, an intentional architecture design is required, which embraces architectural initiatives and guidance for inter-team design and implementation synchronization [7], [9]. The effective evolution of an application's architecture requires the right balance of emergent and intentional architecture design. This balance is an essential determinant for addressing the complexity of large applications architectures and large-scale agile projects [7], [9].

However, literature documenting the influences of scaling agile frameworks on application architectures and the involvement of architects in large-scale agile projects is still scarce. The main objective of this study is to investigate the role of architects in scaling agile frameworks. Based on this objective, three research questions (RQ) were formulated

- *RQ1: What are the types of scaling agile frameworks?*
- *RQ2: What are the roles of architects in scaling agile frameworks?*
- *RQ3: What are the generalizable findings about the role of architects in scaling agile frameworks?*

A. Research methodology

To identify material relevant to stated goal of this study, we applied a structured literature review approach as recommended by Brocke et al. [10]. In the first phase, we defined the scope of the review and identified suitable research questions about the role of architects in scaling agile frameworks. In the second phase, key concepts were identified by concept mapping, which also provided the opportunity to identify additional relevant search terms: *Scaled Agile Organization*, *Enterprise Architecture*, *Scaled Agile Enterprise Architecture*, *Scaled Agile Framework*, *Software Engineering*, and *Scaled Agile Software Engineering*, together with a variety of related concepts, synonyms, and homonyms. These were applied to the subsequent literature search in the third phase. We examined a range of different Information Systems journals, conference proceedings, documentations, and books using EBSCOhost, ScienceDirect, Scopus, ACM Digital Library, IEEExplore, SpringerLink, Emerald Insight,

and Google Scholar. Having compiled the aforementioned list of search terms, we then used them in electronic full-text search queries. In total, we obtained 146 relevant sources. In the fourth phase, we created a concept matrix juxtaposing the different architect roles with the identified scaling agile frameworks to investigate their roles within the scaling agile frameworks. In the last phase, the comparative analysis of architect roles resulted in generalizable findings on the role of architects in scaling agile frameworks.

The remainder of this paper is structured as follows. In Section II, we define large-scale agile developments, providing a primary analysis of the scaling agile frameworks we identified in the literature and describing thoroughly the three most mature frameworks. In Section III, we analyze architect roles identified in scaling agile frameworks. We analyze and discuss these findings in Section IV before concluding the paper with remarks on future research.

II. Scaling Agile Frameworks

The origin of agile ideas in business started with the creation of the Agile Consortium in 1994 [11]. These ideas were discovered independently in software engineering, culminating in the creation of the Agile Manifesto [12] but have exactly the same guiding principles as those in business.

The development of scaling agile frameworks dates back to 1992, with the Crystal Family [13], and has increasingly gained in popularity in the last few years. The main purpose of such frameworks is to manage large agile teams with more than 50 developers distributed across multiple geographical locations in an agile way. Traditional agile methods such as Scrum are not capable of managing this number of developers. However, scaling agile methods introduce new challenges, such as inter-team coordination and distribution of work without a defined architecture or properly defined requirements [14]. We further define the term large-scale agile development and provide a discussion of the frameworks that currently attract the most attention.

A. Definition of large-scale agile development

The difficulty of introducing agile methods increases with organization size [15]. The adoption of agile frameworks often requires changing the entire organizational culture. Larger organizations have more dependencies between projects and teams, which slows down any organizational change. Large size also increases the need for formal documentation, which in turn reduces agility [16]. In addition to inter-team coordination, agile teams also have to interact with other organizational units, which are typically non-agile in nature. Therefore, several companies have devised new approaches to scaling agile methods to projects wherein a lot of people are involved. Hence, the term *large-scale agile development*[1]. refers to agile development in everything from large teams to large multi-team projects that want to make use of agile development principles at the portfolio or enterprise level [18]. Dingsøyr et al. [18] identified that large scale agile projects had been regarded in terms of the number of people or teams, project budget, code base size, or project duration. Examples of cases that were described as *large-scale* covered projects costing over $10 million with teams of more than 50 people, code bases consisting of over half a million lines of code [19], durations of 2 years, and scopes of 60–80 features [20]. Based on their findings, Dingsøyr et al. [18] measured large-scale projects by the number of collaborating and coordinating teams as large-scale projects with 2–9 collaborating teams and very large-scale projects with over 10 collaborating teams.

B. Introduction to the most popular frameworks

The structured literature review described in Section I-A revealed 20 scaling agile frameworks listed in Table I. Most of the frameworks emerged from very basic approaches to agile development, namely XP and Scrum, and were enhanced as necessary in order to be applicable to very large multi-team projects. In Table I, we summarize our findings and provide primary information about the methodologists who invented and published the frameworks, the organizations which were built upon them, and the scaling agile approaches involved. The maturity section of Table I indicates how well-established the particular framework is. Here, we calculate the maturity[2] of a framework based on

- the number of paper contributions that we found in the literature search;
- the number of case studies described on the homepage of the regarded framework;
- the available documentation that could be found either on its homepage or in other sources;
- the training courses and certifications offered by the organization; and
- the content of the community forums and blogs wherein the companies shared their knowledge and other information about the framework.

The LeSS, SAFe, and DA 2.0 frameworks are especially mature, as they are cited very often in the literature, describing many real-world use cases, and also fulfill the other adoption criteria. Therefore, we will subsequently provide more details about these popular frameworks:

LeSS was released in 2008 by Craig Larman and Bas Vodde and extends Scrum with scaling rules and guidelines without losing sight of Scrums's original goals. LeSS specifies additional organizational changes, which are not addressed in traditional Scrum. For instance cross-functional,

[1]A primary systematic literature review on the challenges and success factors for large-scale agile development was conducted by [17].

[2]The maturity is calculated based on the sum of the equally weighted maturity criteria. Particularly, 'Yes' values are coded as '1', whereas 'No' values are coded as '0'. Contributions and cases values of each framework are divided by the max. values of contributions and cases. For instance, the maturity rating of Crystal Family is calculated as follows:
$\frac{17}{35} \cdot 0.2 + \frac{1}{35} \cdot 0.2 + 1 \cdot 0.2 + 0 \cdot 0.2 + 1 \cdot 0.2 = 0.503 \rightarrow \text{◑}$.

Table I
PRIMARY ANALYSIS OF SCALING AGILE FRAMEWORKS.

	Descriptive Information				Maturity					
	Methodologist	Organization	Publication Date	Category	Contri-butions	Cases	Docu-mentation	Training Courses and Certifications	Community, Forum or Blog	Rating
Crystal Family	Alistair Cockburn	-	1992	Set of Methods	17	1	Yes	No	Yes	◑
Dynamic Systems Development Method Agile Project Framework for Scrum	Arie van Bennekum	DSDM Consortium	1994	Framework	28	4	Yes	Yes	Yes	◕
Scrum-of-Scrums	Jeff Sutherland and Ken Schwaber	Scrum Inc.	2001	Mechanism	27	2	Yes	No	Yes	◑
Enterprise Scrum	Mike Beedle	Enterprise Scrum Inc.	2002	Framework	4	-	Yes	Yes	Yes	◑
Agile Software Solution Framework	Asif Qumer and Brian Henderson-Sellers	University of Technology	2007	Framework	2	2	No	No	No	○
Large Scale Scrum	Craig Larman and Bas Vodde	LeSS Company B.V.	2008	Framework	29	22	Yes	Yes	Yes	●
Scaled Agile Framework	Dean Leffingwell	Scaled Agile Inc.	2011	Framework	35	35	Yes	Yes	Yes	●
Disciplined Agile 2.0	Scrott Ambler	Disciplined Agile Consortium	2012	Framework	27	4	Yes	Yes	Yes	●
Spotify Model	Henrik Kniberg, Anders Ivarsson, and Joakim Sundén	Spotify	2012	Model	11	1	Yes	No	Yes	◔
Mega Framework	Rafael Maranzato, Marden Neubert, and Paula Heculano	Universo Online S.A	2012	Framework	2	1	No	No	No	○
Enterprise Agile Delivery and Agile Governance Practice	Erik Marks	AgilePath	2012	Set of Practices	1	-	Yes	No	Yes	◔
Recipes for Agile Governance in the Enterprise	Kevin Thompson	Cprime	2013	Framework	4	1	Yes	Yes	No	◔
Continuous Agile Framework	Andy Singleton	Maxos LLC	2014	Framework	3	-	Yes	No	Yes	◔
Scrum at Scale	Jeff Sutherland and Alex Brown	Scrum Inc.	2014	Framework	9	-	Yes	Yes	Yes	◑
Enterprise Transition Framework	-	agile42	2014	Framework	1	2	Yes	Yes	Yes	◑
ScALeD Agile Lean Development	Peter Beck, Markus Gärtner, Christoph Mathis, Stefan Roock and Andreas Schliep	-	2014	Set of Principles	2	-	Yes	No	Yes	◔
eXponential Simple Continuous Autonomous Learning Ecosystem	Peter Merel	Xscale Alliance	2014	Set of Principles	3	-	Yes	Yes	Yes	◑
Lean Enterprise Agile Framework	-	LeanPitch Technologies	2015	Framework	0	-	Yes	Yes	Yes	◑
Nexus	Ken Schwaber	Scrum.org	2015	Framework	5	-	Yes	Yes	Yes	◑
FAST Agile	Ron Quartel	Cron Technologies	2015	Set of Methods	2	-	Yes	No	Yes	◔

cross-component, end-to-end feature teams are introduced by LeSS through the elimination of traditional team lead and project manager roles [21].

Larman and Vodde proposed two different frameworks depending on the size of the project. The basic LeSS framework provides guidelines and techniques for agile development with less than ten teams [6].

In basic LeSS, a single product owner (PO) is common to all ten teams, but no other special roles are specified compared with standard Scrum. The LeSS framework changes the structure of sprint planning meetings compared with the traditional Scrum approach. Here, each of the agile teams, represented by two members per team, plus the one overall PO decide which chunk of product backlog items to work on. This is in contrast with standard Scrum wherein the rest of the agile team also participates. When contention occurs over a backlog item, the PO mediates between teams. Likewise, sprint review changes to a single meeting for all agile teams. However, it is limited to two team members per agile team. In addition, three more changes are established [6]

- The *inter-team coordination* meeting is designed to increase information sharing and coordination. It can be conducted frequently during the week and can take various forms, including open space, town hall meeting, multi-team daily Scrum, or Scrum of Scrums formats.
- The *joint light product backlog refinement* meeting focuses on refining product backlog items for upcoming sprints. It is restricted to two representatives per team and should not exceed 5% of the sprint duration.
- Finally, a *joint retrospective* is added that aims to identify and plan improvement experiments for the overall product or organization. The PO, scrum master, and one representative from each team attend this meeting.

Agile development with more than ten teams is guided by the second LeSS framework, named LeSS Huge. It introduces additional scaling elements, which are required to manage hundreds of developers in large enterprises. LeSS Huge introduces a new concept, namely, requirement areas (RAs). RAs encompass major areas of customer concern from a product point of view and may grow or shrink over time in order to match product needs. All RAs follow the same sprint cadence and aim for continuous integration across the

entire product. Adding the RA as an attribute in the product backlog creates an area product backlog (APB) view for each RA. This represents a new feature in LeSS Huge. Each product backlog item belongs to one area backlog. Area backlog items are defined, prioritized, and split, as needed, by the area product owner (APO). The APO focuses on one APB and is usually a specialist in that area. The APO acts similar as the PO would in the smaller LeSS framework.

SAFe was released in 2011 by Dean Leffingwell and is now at version 4.0. SAFe builds on existing lean and agile principles that are combined into a method for large-scale agile projects. SAFe provides a soft introduction to the agile world as it specifies many structured patterns. This is often needed for those who are transitioning from a more traditional environment, particularly in the context of a large project. A common problem with agile adoption is the difficulty in introducing such a major cultural change to an organization. Thus, SAFe provides the structure needed to make the transition more predictable, even though it follows agile practices and stresses autonomy and decision-making for knowledge workers. SAFe highlights four levels of organization: team, program, value stream, and portfolio. Each level integrates agile and lean practices, manages its own activities, and is aligned with the other levels.

At the team level, the techniques outlined are those used in Scrum, and two-week sprint cycles are recommended. Each team comprises 5–9 members and has a scrum master and a PO, similar to standard Scrum. All SAFe teams are part of one agile release train (ART), a team of agile teams that delivers a continuous flow of incremental releases of value. Each agile team is responsible for defining, building, and testing stories from its team backlog in a series of iterations using common iteration cadences and synchronization to align its activities with other teams so that the entire system is iterating in unison. Teams use ScrumXP or Kanban to deliver prototypes every two weeks [22].

At the program level, SAFe extends Scrum using the same ideas but on a higher level. The program level is based on an ART, which is composed of five sprint cycles. There is also a sixth innovation planning sprint, which allows teams to innovate, inspect, and adapt. Teams, roles, and activities are organized around the ART [22]. At this level, a product manager (PM) serves as the content authority for the ART and is accountable for identifying program backlog priorities. In addition, the PM works with POs to optimize feature delivery and direct the work of POs at the team level. A release train engineer (RTE) facilitates program level processes and execution, escalates impediments, manages risk, and helps to drive continuous improvement.

At the optional value stream level, the value stream engineer (VSE) plays a similar role, facilitating and guiding the work of all ARTs and suppliers. Further important roles, such as business owner, DevOps team member, release manager, and solution manager have been described in [22].

SAFe also specifies processes at one higher level, the portfolio level, using lean principles, such as optimizing value streams, which are long-lived series of steps used to deliver value. These help executives and leaders identify and prioritize epics and features that can be broken down at the program level and scheduled for ARTs [22].

The **DA 2.0** framework, previously kwown as Disciplined Agile Delivery, was released in 2012 by Scott Ambler and Mark Lines [23]. In comparison with SAFe, DA 2.0 aims to address areas that are not thoroughly covered in smaller scaling agile frameworks and recommends three phases: inception, construction, and transition. While many agile frameworks address what DA 2.0 calls the construction phase, DA 2.0 provides recommendations for processes that come both earlier in the project (inception) and as teams prepare for delivery (transition). DA 2.0 also provides flexibility by suggesting different process guidelines for four categories of life cycles: agile/basic, lean/advanced, continuous delivery, and exploratory.

The construction phase of agile/basic is Scrum, whereas the lean/advanced life cycle uses processes similar to Kanban. The inception phase is used to stock a work item pool that is organized to achieve business values, fixed delivery dates, expedited delivery, or some other intangible goal. During the transition phase, planning, retrospection, prototyping, stand up meetings, and other activities are undertaken. The continuous delivery life cycle focuses on mature DevOps, continuous integration, and deployment processes for projects that require frequent delivery to stakeholders. The exploratory life cycle minimizes early planning in favor of fast delivery, gaining feedback, and incorporating that feedback into the next delivery [23].

In the third continuous delivery life cycle, the inception phase is explicit and has a very brief transition period. In this life cycle, products are produced on a very regular basis: daily, weekly, or monthly [23].

The last life cycle, the exploratory life cycle, aims to encourage agile teams to put themselves in start-up or research situations wherein the stakeholders have clear ideas for a new product but do not yet understand the needs of their user base [23].

Table II
IDENTIFIED ARCHITECT ROLES IN SCALING AGILE FRAMEWORKS.

	Enterprise Architect	Software Architect	Solution Architect	Information Architect
DSDM	-	X	X	-
SAFe	X	X	X	X
DA 2.0	X	X	X	-
EADAGP	X	-	-	X

III. The Role of Architects in Scaling Agile Frameworks

Since traditional agile methods, such as XP or Scrum, do not include the role of architects, we recognize that this is no longer valid for scaling agile frameworks. In particular, we have seen that several scaling agile frameworks involve various architect roles. We have selected a set of predominant architect roles relevant to the realm of enterprise architecture (EA) from refs. [24] and [25] to describe their function in scaling agile frameworks. These roles comprise the *enterprise architect*, *software architect*, *solution architect*, and *information architect*[3]. We have searched each relevant source for these architect roles and related them to the different scaling agile frameworks. The results are summarized in Table II.

Only 4 out of 20 scaling agile frameworks include architect roles, namely Dynamic Systems Development Method Agile Project Framework for Scrum (DSDM), SAFe, DA 2.0, and Enterprise Agile Delivery and Agile Governance Practice (EADAGP). The findings also reveal that no framework describes all architect roles in detail. The frameworks predominantly consider the roles of enterprise, software, and solution architects. In addition, the results show that more mature frameworks are more likely than others to describe architect roles. However, due to limited information and unavailable documentation, further analyses are required to emphasize this assumption.

We have created the following role description template, based on refs. [24], [26], [27] to describe architect roles

- *Key concerns* describe the key interests of the architect role.
- *Area of interests* describes the area of interests of the architect role, e.g., special project tasks.
- *Contributions* describe the contributions of the architect role, e.g., resources like work, financial capital, or other types of engagements.
- *Strategies* describes the appropriate strategies for working with the architect role from management view.
- *Responsibilities* describes the responsibilities of the architect role.
- *Commitments* describes the commitments of the architect role, which can be either active opposition, passive opposition, neutral, passive support, or active support.

The following sections present our results about the role of architects in various scaling agile frameworks.

A. The role of enterprise architect

The role of enterprise architect is only considered by SAFe, DA 2.0, and EADAGP.

In **SAFe**, an enterprise architect works with business stakeholders and software and solution architects to drive holistic technology implementation across value streams.
The enterprise architect is *concerned* with driving EA strategy, which comprises five key aspects, namely choice of technology, software and solution strategy, development and deployment infrastructure strategy, inter-program collaboration, and implementation strategy. This is communicated, along with other key business drivers of architecture, to system architects and nontechnical stakeholders.
The main *contributions* of the enterprise architect are providing strategic technical directions and driving collaboration of programs and teams around a common technical vision.
The portfolio level represents the enterprise architect's *area of interest*.
The *strategy* for working with enterprise architects is to involve them actively in the portfolio level by ensuring the presence of enterprise-wide architectural systems, platforms, and infrastructures.
The key *responsibilities* of the enterprise architect are

- maintaining a high-level and holistic vision of enterprise solutions and development initiatives;
- understanding and communicating strategic themes and other key business drivers for architecture to system architects and nontechnical stakeholders;
- working with business stakeholders and software and solution architects to drive holistic technology implementation across value streams;
- working closely with software and solution architects to ensure that individual program and product strategies align with enterprise objectives;
- participating in the strategy for building and maintaining the enterprise architectural runway; and
- facilitating the reuse of ideas, components, and patterns.

The *commitment* of the enterprise architect is the active support of agile teams [22].

Enterprise awareness is one of the key aspects of the **DA 2.0** framework. Enterprise awareness motivates agile teams to consider the overall needs of the organization and to leverage existing assets in alignment with an enterprise-level strategy. DA 2.0 recommends that agile teams work closely with enterprise professionals, such as enterprise architects.
Within DA 2.0, the enterprise architect has both a primary role as a stakeholder and a secondary role as a specialist for assisting agile teams. The *key concerns* of enterprise architects are addressing strategies for supporting delivery teams and other stakeholders, evolving and capturing the EA, and governing the EA efforts.
The inception, construction, and ongoing process goals of the DA 2.0 framework form the *area of interest* of the enterprise architect. In the inception phase, the enterprise architect works closely with agile teams to align them with enterprise goals and to provide non-functional requirements (NFRs). Further, the enterprise architect supports agile teams during the initial architectural envisioning and modeling efforts and

[3]We have also considered synonyms for each architect role, e.g., domain architect denotes the same role as solution architect.

the initial technical strategy definition phase by ensuring that they leverage as much of the existing infrastructure as possible. In the construction phase, the enterprise architect collaborates with agile teams to ensure that their solution reflects the overall strategy of the organization. Within the ongoing process goal, the enterprise architect holds regular coordination meetings with product management teams to ensure consistency and manage dependencies across teams.
The enterprise architect *contributes* to software development by providing guidance to agile teams by producing high-level technology and business roadmaps, which capture the organization's vision and by helping agile teams to understand the overall vision. The *strategy* for working with enterprise architects is to involve them passively in development projects.
The key *responsibilities* of the enterprise architect are

- supporting and collaborating closely with stakeholders on a regular basis to understand their needs and to develop the organization's roadmap;
- supporting and collaborating closely with agile teams on a regular basis to guide them through the business and technical roadmaps and help them to identify potentially reusable assets and technical debts;
- negotiating technical dependencies between solutions;
- exploring architectural views; and
- adopting and tailoring architectural frameworks.

The *commitment* of the enterprise architect is the passive support of agile teams by providing guidance and roadmaps [23], [28], [29], [30], [31], [32].

The **EADAGP** framework includes an event-driven governance model, which provides a lightweight, lean, and virtual governance model design. EADAGP introduces a governance buffer zone, which aims to protect agile teams from the slowness, friction, and rigidity of traditional IT governance models. The agile governance buffer zone is a subtractive layer that is realized by combining three different styles of governance, namely top-down prescriptive governance (traditional IT governance), community governance, and self-governance. The enterprise architect forms, along with scrum master(s), PO(s), and the agile governance owner the agile community team (ACT). This is a community governance construct, responsible for governing multiple agile teams that are aligned with one or more releases.
Within the ACT, the enterprise architect is *concerned* with governance requirements that span multiple sprints or releases and cross-sprint team coordination and collaboration. Community governance, in particular the ACT, represents the *area of interest* of the enterprise architect.
The main *contributions* of the enterprise architect are addressing governance requests, escalating them to IT governance if they cannot be addressed, and notifying agile teams and IT governance for their agreement and/or approval.
The *strategy* for working with enterprise architects is to involve them passively within the ACT so they can make appropriate governance decisions and provide guidance.
The key *responsibilities* of the enterprise architect are

- escalating governance issues to IT governance;
- supporting agile teams by making appropriate governance decisions and providing guidance;
- communicating governance decisions to enterprise IT governance for their agreement;
- supporting the governance backlog grooming process;
- harmonizing governance requirements across sprints and agile teams;
- reporting technology and architecture requirements/issues to EA oversight for alignment and issue resolution;
- identifying security requirements and challenges that may not have been pre-determined; and
- raising potential compliance and risk requirements that have to be reviewed and signed off by governance, risk and compliance bodies, and EA sign offs.

The *commitment* of the enterprise architect is the passive support of agile teams by protecting them from the slowness and rigidity of traditional IT governance [33], [34].

B. The role of software architect

DSDM, SAFe, and DA 2.0 include the role of software architects.

DSDM does not explicitly prescribe the role of software architect but it does specify the role of a technical coordinator, whose responsibilities can be allocated to more than one person, e.g., a system architect. Thus, we will concentrate on describing the role of the technical coordinator, which is relevant to the role of a software architect.
As the project's technical authority, the software architect's *key concern* is to ensure that the project is technically coherent and meets the desired technical standards. The software architect holds the business and technical visions for the project.
The project level constitutes the *area of interest* of the software architect. Therein, the software architect may be a part of a project board or steering committee for the project. Technical development and direction of the solution and system architecture definitions are the software architect's main *contributions*.
The *strategy* for working with software architects is to involve them as technical coordinators.
The *responsibilities* of the software architect are

- agreeing and controlling the technical architecture;
- identifying and owning architectural and other technically based risks;
- working with business analysts to evaluate the technical options and decide on the best way to turn the high-level business requirements into a technical solution;
- promoting standards for technical best practice;

- acting as the final arbiter of technical differences between agile team members; and
- defining the system architecture that constitutes the technical framework within which the solution will be developed and providing a high-level description of the structure of that solution.

The *commitment* of the software architect is passive support of agile teams [11], [35], [36].

SAFe's key roles at the program level include the RTE, product management, and system architect/engineer, hereafter the software architect. The software architect role is filled by an individual or small team that has technical responsibility for the overall architectural and engineering design of the system and aligns ARTs with the common technical and architectural vision for the solution under development. The software architect participates in defining the system, as well as any subsystems and interfaces, validating technology assumptions, and evaluating alternatives. The software architect works at a higher level of abstraction than the teams and supports system development by providing, communicating, and evolving the larger technological and architectural view of the solution.

The system architect is *concerned* with executing the upfront architecture design, guiding the emergent architecture for all program teams, and defining the architectural runway that supports new feature development, as well as providing guidance for common solution behaviors, shared components, and separation of concerns.

The program level, particularly the ART, represents the software architect's main *area of interest*.

The software architect *contributes* to software development by working with agile teams and providing technical enablement with respect to subsystems and capability areas under the purview of the ART.

The *strategy* for working with software architects is to involve them at program level.

The key *responsibilities* of the software architect are

- defining NFRs, major system elements, subsystems, and interfaces;
- preparing the architecture vision briefing within the program increment (PI) planning event;
- presenting the architecture vision, which may include descriptions of new architectural epics for common infrastructure, any large-scale refactors under consideration, and system-level NFRs; and
- supporting the PO by refining the team backlog.

In addition, the software architect shares several common responsibilities with the solution architect, which are

- defining and refining NFRs, subsystems, and interfaces to ensure that the solution meets relevant standards and other system quality requirements;
- defining subsystems and their interfaces, allocating responsibilities to subsystems, understanding solution deployment, and communicating requirements for interactions with the solution context;
- preparing for the PI planning event by updating enabler definitions and models;
- assisting with decision-making and sequencing of the key technological infrastructures that will host the new business functionality;
- creating and supporting enabler epics by steering them through the Kanban system, providing both the guidance needed to analyze them and the information needed to estimate and implement them;
- working with portfolio stakeholders, particularly the enterprise architect, to develop, analyze, split, and realize the implementation of enabler epics; and
- planning and developing the architectural runway in support of upcoming business features and capabilities.

The *commitment* of the software architect is the active support of agile teams within the program level [22].

As already mentioned in Section II-B, **DA 2.0** introduces the AM role of architecture owner, which is typically the software or solution architect, hereafter the software architect[4]. The software architect is *concerned* with owning architecture decisions for the team and facilitating the creation and evolution of the overall solution design.

The first two phases, namely the inception and transition phases, as well as the ongoing process goals that occur throughout the delivery life cycle represent the *area of interest* of the software architect. The software architect is involved during the inception phase by contributing to exploration of the initial solution requirements, alignment of the solution with both the business and technical directions of the organization, and identification of the initial technical strategy. During the construction phase of the solution, the software architect updates the architectural handbook in the iteration in which the features are delivered. The software architect contributes to the goal of activity coordination within the ongoing process goal by having regularly scheduled information meetings with the PO to share information about work details, priorities, dependencies, and issues. The software architect mainly *contributes* to the identification of the initial technical strategy, definition of the architecture, and development of technical aspects of the overall solution architecture. The *strategy* for working with software architects is to involve them during the two first life cycle phases and with ongoing process goals of DA 2.0.

The key *responsibilities* of the software architect are

- guiding the creation and evolution of the solution architecture that the team is working on;
- mentoring and coaching other team members with best practices of architecture;
- understanding the architectural direction and standards

[4]Since DA 2.0 does not differentiates between the roles of software architect and solution architect, we have merged the findings for both roles.

of the organization and helping to ensure that the agile team adheres to them appropriately;
- understanding existing enterprise assets such as frameworks, patterns, and subsystems and ensuring that the team uses them where appropriate;
- ensuring that the solution will be easy to support by encouraging good design and refactoring to minimize technical debt;
- ensuring that the solution is integrated and tested on a regular basis, ideally via continuous integration;
- working closely with the team lead to identify technical risks in the project and determining strategies to mitigate them; and
- leading the initial architecture envisioning effort at the beginning of the project and supporting the initial requirements envisioning effort.

The *commitment* of the software architect is the active support of agile teams [23], [31], [32].

C. The role of solution architect

The role of solution architects is supported by DSDM, SAFe, and DA 2.0.

As discussed previously in Section III-B, **DSDM** does not include the architect roles explicitly. However, a solution architect can also be allocated to the role of a technical coordinator. Thus, we will describe the technical coordinator characteristics that are relevant to the solution architect role.
The solution architect is *concerned* with definition of the solution architecture.
Thus, the solution architecture represents the solution architect's *area of interest*. Within DSDM, the solution architecture is set during the foundation phase, which makes a preliminary investigation of the feasibility of the solution and establishes a fundamental understanding of the business rationale for the project and the potential solution. The solution architecture constitutes an evolutionary product, which provides a high-level design framework for the solution. It covers both business and technical aspects of the solution.
The main *contribution* of the solution architect is in the definition of the solution architecture during the foundation phase of the project.
The *strategy* for working with solution architects is to involve them actively in defining the solution architecture during the feasibility, foundation, evolutionary development, and deployment phases.
The solution architect is *responsible* for the overall design and integrity of the technical aspects of the solution, which comprise the solution architecture.
The *commitment* of the solution architect is the active support of agile teams by providing the definition of the solution architecture [11], [35], [36].

At the value stream level, **SAFe** introduces additional roles, namely solution management, VSE, and solution architect/engineer, hereafter solution architect. The role of solution architect is filled by cross-disciplinary teams that take a systemwide view of solution development.
The solution architect is *concerned* with the overall architectural design of the solution, definition of the higher-level functional and NFRs, determination of major components and subsystems, and definition of interfaces.
The primary *area of interest* of the solution architect is the value stream level.
As a technical leader, the solution architect *contributes* to the entire solution under development by communicating and evolving the larger technological and architectural view of the solution and aligning the value stream and ARTs to a common technological and architectural vision.
The *strategy* for working with solution architects is to involve them actively and to enable their close collaboration with business stakeholders, teams, customers, suppliers, and third-party stakeholders.
Since the solution architect shares several common responsibilities with the software architect in SAFe, we will only describe responsibilities, which are exclusive to the solution architect (see Section III-B for common responsibilities).
The exclusive *responsibilities* of the solution architect are
- supporting the solution management by managing the value stream kanban;
- discussing upcoming enabler capabilities and epics with the solution management;
- defining the overarching architecture that connects the solution across ARTs;
- working with the system architect to guide the architecture developed by the ARTs;
- ensuring technical alignment with the solution context, including interfaces and constraints;
- attending to the value stream and ART PI planning events; and
- updating progress toward milestones, program PI objectives, and dependencies among the ARTs

The *commitment* of the solution architect is the active support of agile teams at value stream level [22].

D. The role of information architect

The role of information architect is only supported by SAFe and EADAGP.

Within **SAFe**, the *key concern* of the information architect is to participate in a shared services role in order to support development by quickly bringing specialized expertise to bear on areas of the system or solution that require unique knowledge and skills.
The *area of interest* of the information architect is the ART and value stream.
The main *contribution* of the information architect is to support agile teams on the ART and to contribute to the architectural runway. Along with a system architect, a scrum master and one or two agile teams. They create the technological infrastructure to support the highest-priority features

in a near-term PI and the intentional architecture that guides cross-team design and implementation synchronization.
The *strategy* for working with information architects is to involve them actively in the project and to embed them periodically in agile teams.
The *responsibilities* of the information architect are

- participating in PI planning as well as in pre- and post-PI planning;
- driving requirements and taking ownership of dependent backlog items;
- collaborating with agile teams to fulfill dependencies during PI executions; and
- participating in system and solution demos.

The *commitment* of the information architect is the active support of agile teams [22].

Within the **EADAGP** framework, the *key concerns* of the information architect is participation within the ACT. The information architect works on governance requirements that cut cross multiple sprint teams or releases.
The *area of interest* of the information architect is the ACT.
The main *contributions* of the information architect is providing information architecture governance requirements that span multiple sprints or releases.
The *strategy* for working with information architects is to involve them passively in the project.
The key *responsibilities* of the information architect are

- escalating and communicating governance issues and decisions to IT governance;
- supporting agile teams by making appropriate governance decisions and providing guidance;
- supporting the governance backlog grooming process;
- harmonizing governance requirements across sprints and agile teams;
- surfacing technology and architecture requirements;
- identifying security requirements and challenges that may not have been pre-determined; and
- raising potential compliance and risk issues.

The *commitment* of the information architect is the passive support of agile teams and, thus, of software development since the information architect is only responsible for imposing governance requirements on agile teams [33].

IV. Discussion

We now discuss the main outcomes of our findings. ***Increasing development speed by balancing emergent and intentional architecture design.*** While some scaling agile frameworks, e.g., LeSS, are against upfront architecture design, other frameworks, such as DA 2.0 and SAFe, endorse upfront architecture design and planning. While they highlight the dangers of traditional architectural habits, e.g., heavyweight documentations, tedious upfront design approaches, and imposed architectural guidelines, they also realize that some initial envisioning performed at the beginning of the project can increase effectiveness and reduce excessive redesign efforts as the teams are steered in the intended direction. Only a close collaboration between agile teams, software, solution, and enterprise architects can enable an optimal interplay of emergent design and intentional architecture.

Finding the right balance between centralized and decentralized architectural decision-making. Escalating any type of architectural decision to higher levels of authority increases delay and decreases the usefulness of the decision. A balanced combination of centralized and decentralized decision-making provides many benefits, e.g., faster time to market and higher-quality products and services.

Sparing agile development from traditional IT governance. Some scaling agile frameworks, such as DSDM and DA 2.0, recognize the real value of agility in terms of project productivity and solution quality while acknowledging and accepting necessary constraints, e.g., governance, architecture, and infrastructure strategies, which often exist when working in a corporate environment. However, traditional governance models are slowing down agile teams in large organizations. SAFe, EADAGP, and DA 2.0 recommend new governance models that are collaborative, decentralized, and light-weight. These new models allow teams to decentralize their decision making and to govern themselves.

Ensuring the reuse of enterprise assets. Architects are aware of existing enterprise assets, e.g., patterns and standards, which are available for reuse, and ensure that agile teams utilize them where applicable. This accelerates the development process and reduces time to market.

V. Conclusion and future work

In this study, we have motivated the need for scaling existing agile methods to large-scale agile development due to their deficiencies in inter-team coordination and communication. We have then presented a primary analysis of the identified scaling agile frameworks. Based on our maturity assessment, we have identified LeSS, SAFe, and DA 2.0 as the most mature frameworks. Finally, we have extensively characterized the different architect roles that have been identified in scaling agile frameworks. Our findings indicate that architects both actively and passively support agile teams by driving architectural initiatives, participating in architectural runways, harmonizing governance requirements, and ensuring technical alignment in solution contexts. Future research may analyze the challenges faced by architects in scaling agile environments by conducting case studies in organizations that can provide practical experience of adopting scaling agile frameworks.

References

[1] P. Weill and S. Woerner, "Thriving in an increasingly digital ecosystem," *MIT Sloan Management Review*, vol. 56, no. 4, p. 27, 2015.

[2] B. Sherehiy, W. Karwowski, and J. Layer, "A review of enterprise agility: Concepts, frameworks, and attributes," *International Journal of industrial ergonomics*, vol. 37, no. 5, pp. 445–460, 2007.

[3] E. Overby, A. Bharadwaj, and V. Sambamurthy, "Enterprise agility and the enabling role of information technology," *European Journal of Information Systems*, vol. 15, no. 2, pp. 120–131, 2006.

[4] P. Kettunen, "Extending software project agility with new product development enterprise agility," *Software Process: Improvement and Practice*, vol. 12, no. 6, pp. 541–548, 2007.

[5] M. Alqudah and R. Razali, "A review of scaling agile methods in large software development," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 6, no. 6, pp. 28–35, 2016.

[6] A. Vaidya, "Does dad know best, is it better to do less or just be safe? adapting scaling agile practices into the enterprise," *PNSQC. ORG*, pp. 828–837, 2014.

[7] "Agile architecture," http://www.scaledagileframework.com/agile-architecture/, accessed: 2017-04-26.

[8] M. Mocker, "What is complex about 273 applications? untangling application architecture complexity in a case of european investment banking," in *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*. IEEE, 2009, pp. 1–14.

[9] M. Waterman, "Reconciling agility and architecture: A theory of agile architecture," Ph.D. dissertation, Victoria University of Wellington, 2014.

[10] J. Vom Brocke, A. Simons, B. Niehaves, K. Riemer, R. Plattfaut, and A. Cleven, "Reconstructing the giant: On the importance of rigour in documenting the literature search process," in *ECIS*, vol. 9, 2009, pp. 2206–2217.

[11] A. B. Consortium, *The DSDM Agile Project Framework Handbook*. Agile Business Consortium, 2014.

[12] M. Fowler and J. Highsmith, "The agile manifesto," *Software Development*, vol. 9, no. 8, pp. 28–35, 2001.

[13] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile software development methods: review and analysis," VTT Technical report, Tech. Rep., 2002.

[14] D. Leffingwell, *Scaling Software Agility: Best Practices for Large Enterprises (The Agile Software Development Series)*. Addison-Wesley Professional, 2007.

[15] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Inf. Softw. Technol.*, vol. 50, no. 9-10, pp. 833–859, Aug. 2008.

[16] M. Lindvall, D. Muthig, A. Dagnino, C. Wallin, M. Stupperich, D. Kiefer, J. May, and T. Kahkonen, "Agile software development in large organizations," *Computer*, vol. 37, no. 12, pp. 26–34, 2004.

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

[18] T. Dingsøyr and N. Moe, *Towards Principles of Large-Scale Agile Development*. Cham: Springer International Publishing, 2014, pp. 1–8.

[19] K. Petersen and C. Wohlin, "The effect of moving from a plan-driven to an incremental software development approach with agile practices," *Empirical Softw. Engg.*, vol. 15, no. 6, pp. 654–693, Dec. 2010.

[20] E. Bjarnason, K. Wnuk, and B. Regnell, "A case study on benefits and side-effects of agile practices in large-scale requirements engineering," in *Proceedings of the 1st Workshop on Agile Requirements Engineering*, ser. AREW '11. New York, NY, USA: ACM, 2011, pp. 31–35.

[21] B. V. Craig Larman, "Scaling agile development," *CrossTalk*, pp. 8–12, 2013.

[22] D. Leffingwell, A. Yakyma, R. Knaster, D. Jemilo, and I. Oren, *SAFe® 4.0 Reference Guide: Scaled Agile Framework® for Lean Software and Systems Engineering*. Addison-Wesley Professional, 2016.

[23] S. Ambler and M. Lines, *Disciplined agile delivery: A practitioner's guide to agile software delivery in the enterprise*. IBM Press, 2012.

[24] V. Haren, *TOGAF Version 9.1*. Van Haren Publishing, 2011.

[25] P. A. Khosroshahi, M. Hauder, A. Schneider, and F. Matthes, "Enterprise architecture management pattern catalog version 2.0," Technische Universität München, Tech. Rep., 2015.

[26] E. Andersen, K. Grude, and T. Haug, *Goal directed project management: effective techniques and strategies*. Kogan Page Publishers, 2009.

[27] J. R. Turner, *People in project management*. Gower Publishing Company, 2003.

[28] S. Ambler and M. Lines, "Scaling agile software development: Disciplined agility at scale," Disciplined Agile Consortium, Tech. Rep., 2014.

[29] ——, "The disciplined agile process decision framework," in *International Conference on Software Quality*. Springer, 2016, pp. 3–14.

[30] ——, "Going beyond scrum disciplined agile delivery," Disciplined Agile Consortium, Tech. Rep., 2013.

[31] ——, "Scaling agile software development tactically: Disciplined agile delivery at scale," Disciplined Agile Consortium, Tech. Rep., 2016.

[32] "Disciplined agile 2.x: A process decision framework," http://www.disciplinedagiledelivery.com/, accessed: 2017-04-26.

[33] E. Marks, "Governing enterprise agile development without slowing it down: Achieving friction-free scaled agile governance via event- driven governance," AgilePath Corporation, Tech. Rep., 2014.

[34] ——, "A lean non-functional requirements (nfr) framework: A common framework for governance, risk and compliance as well as traditional nfrs," AgilePath Corporation, Tech. Rep., 2017.

[35] A. B. Consortium, *DSDM Atern Handbook*. Agile Business Consortium, 2008.

[36] A. Craddock, K. Richards, D. Tudor, B. Roberts, and J. Godwin, "The dsdm agile project framework for scrum," DSDM Consortium, Tech. Rep., 2012.

Evolution of the Agile Scaling Frameworks

Ömer Uludağ[1(✉)], Abheeshta Putta[2], Maria Paasivaara[2,3], and Florian Matthes[1]

[1] Technische Universität München, München, Germany
{oemer.uludag,matthes}@tum.de
[2] Aalto University, Espoo, Finland
{abheeshta.putta,maria.paasivaara}@aalto.fi
[3] LUT University, Lappeenranta, Finland
maria.paasivaara@lut.fi

Abstract. Over the past decade, agile methods have become the favored choice for projects undertaken in rapidly changing environments. The success of agile methods in small, co-located projects has inspired companies to apply them in larger projects. Agile scaling frameworks, such as Large Scale Scrum and Scaled Agile Framework, have been invented by practitioners to scale agile to large projects and organizations. Given the importance of agile scaling frameworks, research on those frameworks is still limited. This paper presents our findings from an empirical survey answered by the methodologists of 15 agile scaling frameworks. We explored (i) framework evolution, (ii) main reasons behind their creation, (iii) benefits, and (iv) challenges of adopting these frameworks. The most common reasons behind creating the frameworks were improving the organization's agility and collaboration between agile teams. The most commonly claimed benefits included enabling frequent deliveries and enhancing employee satisfaction, motivation, and engagement. The most mentioned challenges were using frameworks as cooking recipes instead of focusing on changing people's culture and mindset.

Keywords: Agile scaling frameworks · Large-scale agile · Survey

1 Introduction

Ever since the creation of the Agile Manifesto in 2001, practitioners and academics have devoted a great deal of attention to agile software development methods [1]. Initially, they were designed for small, co-located, and self-organizing teams that develop software in close collaboration with business customers using short iterations [2]. Hence, agile methods have been primarily applied to projects within the so-called *'agile sweet spot'*, i.e., small and co-located teams of less than 50 persons with easy access to the user and business

P. Gregory et al. (Eds.): XP 2021, LNBIP 419, pp. 123–139, 2021.
https://doi.org/10.1007/978-3-030-78098-2_8

experts and that develop non-life-critical software [3]. Given the successful adoption of agile methods in small organizations and projects, also many large software organizations have begun to adopt these methods [4]. However, the adoption of agile methods outside the agile sweet spot poses significant challenges to organizations, such as coordination challenges in multi-team environments [5]. To resolve issues associated with the adoption of agile methods in large-scale organizations and projects, several agile scaling frameworks, such as Large Scale Scrum (LeSS)[1] and Scaled Agile Framework (SAFe)[2], have been created both by some custodians of existing agile methods and by others who have worked with companies to scale agile methods to their settings [4,6,7]. As large organizations face growing pressures and expectations to become more agile, and the agile scaling frameworks claim to provide off-the-shelf solutions to scaling, their adoption has rapidly increased in industry, as confirmed by the yearly non-scientific survey on the state of agile development conducted by VersionOne [6–8].

Not only is there a growing interest in adopting agile scaling frameworks from an industrial perspective [8], but there is also a growing academic interest to study the adoption of these frameworks [6]. A systematic mapping study by Uludağ et al. [6] uncovered the topic of agile scaling frameworks as a major research stream in the field of large-scale agile development, with a total of 16% of all published studies related to large-scale agile development. The existing literature on the scaling frameworks mainly investigates how individual frameworks are adopted based on case studies (cf. [9]) followed by a comparison of the frameworks based on their underlying characteristics based on literature reviews (cf. [10]). However, the existing literature on agile scaling frameworks disregards the following topics: (i) providing a comprehensive overview of agile scaling frameworks and their evolution, (ii) studying the reasons behind creating these frameworks, and (iii) investigating the benefits and (iv) challenges of adopting these frameworks. To address this research gap, we conducted a survey with the creators/methodologists of known agile scaling frameworks and aim to answer the following research questions (RQs):

- *RQ1: How did the agile scaling frameworks evolve over the years?*
- *RQ2: What are key reasons behind creating of agile scaling frameworks?*
- *RQ3: What are the claimed benefits of adopting agile scaling frameworks?*
- *RQ4: What are the claimed challenges of adopting of agile scaling frameworks?*

The remainder of this paper is structured as follows. In Sect. 2, we provide an overview of related work. In Sect. 3, we portray the research design of our paper. Section 4 presents the result of our survey. In Sect. 5, we discuss our main findings and limitations and conclude our study with a summary of our results and remarks on future research.

[1] https://less.works/, last accessed on: 03-10-2021.
[2] https://www.scaledagileframework.com/, last accessed on: 03-10-2021.

2 Background and Related Work

The successful adoption of agile methods in small teams ignited a new passion among firms to start using agile methods in large projects, even beyond software development, across the enterprise [11]. This phenomenon is often referred as *'large-scale agile development'* [12]. In line with Dikert et al. [5], we understand the term *'large-scale agile development'*, as the application of agile methods in large multi-team settings consisting of 50 persons or more, or at least six teams.

Over the past two decades, software engineers and researchers have devoted a great deal of attention to agile software development [13]. Within few years, various agile methods appeared on the landscape, such as Extreme Programming and Scrum, to name a few [1]. Figure 1 presents the various agile methods, their interrelationships, and their evolutionary paths [13].

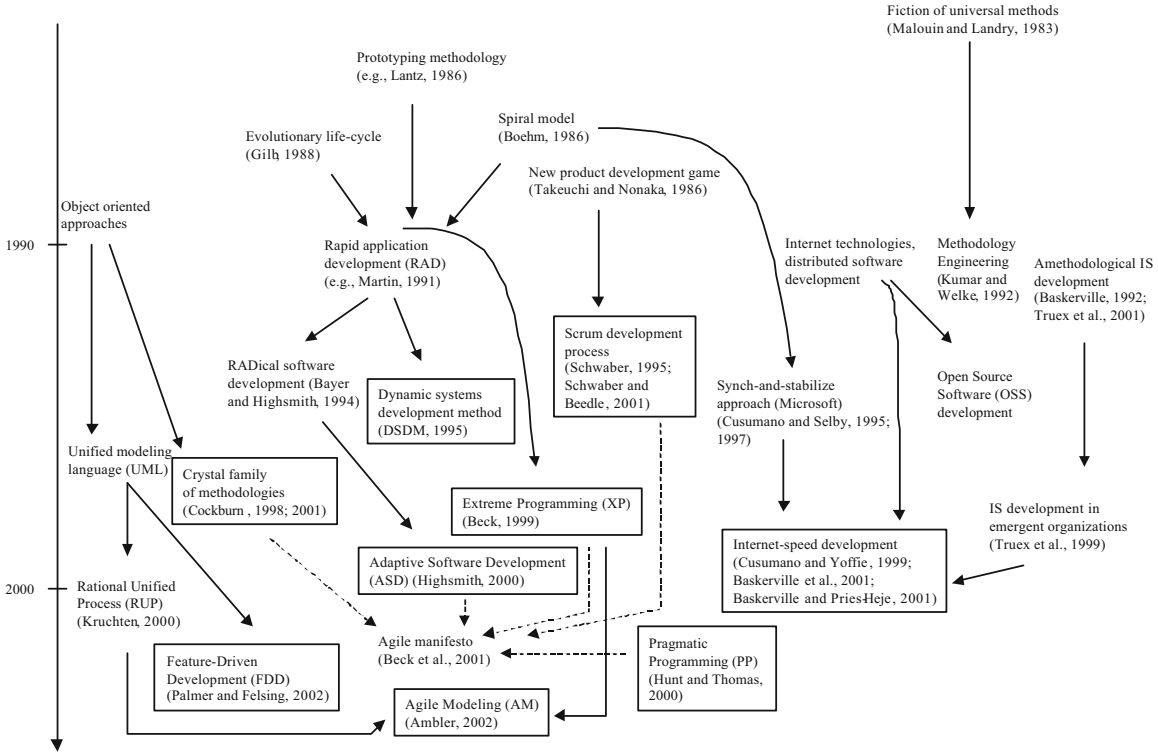


Fig. 1. Evolutionary map of agile methods [13]

Agile methods adhering to varying degrees to the tenets of the Agile Manifesto[3] share some common characteristics, e.g., iterative and incremental development and focus on small releases [1]. The ideal context of applying agile methods in software projects lies within the so-called *'agile sweet spot'*, i.e., small and co-located teams of less than 50 persons with easy access to the user and business experts and that develop non-life-critical software [3]. However, applying agile methods both for larger projects or in larger companies [5], i.e., scaling agile

[3] http://agilemanifesto.org/, last accessed on: 03-10-2021.

methods, involves two significant challenges. First, the scaling of agile methods entails additional scaling and complexity factors that summon *'bitter spot'* conditions for agile methods, such as a large number of teams, geographical distribution, entrenched culture, or formal governance structures [14]. Second, present agile methods do not provide sufficient guidance on dealing with these scaling and complexity factors [15]. Thus, custodians of existing agile methods and consultants that have worked with companies in scaling agile to their settings have proposed several agile scaling frameworks over the last years to address the limitations of the agile methods in large organizations and projects [6,7,10]. These frameworks incorporate predefined workflow patterns to deal with issues related to large number of teams, inter-team coordination, and customer involvement [10,16].

Due to the importance of this topic to companies, researchers have started to study the frameworks' adoption [6]. Based on a structured literature review, Uludağ et al. [17] identified 20 different agile scaling frameworks presented in Table 1[4]. Secondary studies on the scaling frameworks compare some of them based on different criteria. For instance, Alqudah and Razali [10] juxtapose *DAD*, *LeSS*, *Nexus*, *RAGE*, *SAFe*, and *Spotify* based on, e.g., team size, available training and certificates, and the underlying agile methods and practices. Diebold et al. [18] provide a map visualizing underlying agile practices of different frameworks, such as *DAD*, *LeSS*, and *Nexus*, to support organizations in the selection of appropriate frameworks. Based on 13 agile transformation cases, Conboy and Carroll [16] provide nine challenges and a set of recommendations associated with agile scaling frameworks, such as *LeSS*, *Nexus*, *S@S*, and *Spotify*.

Although agile scaling frameworks have received some attention from academics [6], to the best of our knowledge, there is no other work that provides an overview of agile scaling frameworks, their evolution, and reasons, as well as the benefits and challenges of these frameworks.

3 Research Methodology

Survey Design. To answer the research questions, we created a survey following the guidelines suggested by Linåker et al. [19]. We opted to conduct a survey as it often aims to provide a state-of-the-art overview on particular methods [20], such as agile scaling frameworks. As a large part of our survey consists of closed-ended questions to quantitatively analyze the agile scaling frameworks, we used a survey as it is a suitable means to provide a quantitative description of the data [20]. The questionnaire consisted of four sections with a total of 22 questions[5]. The first section included questions on the framework background, e.g., reasons

[4] We extended the table by Uludağ et al. [17] by adding a column to show the scaling levels of the frameworks and expanded the list of the frameworks by two additional frameworks: *HSD* and *Parallel* as their methodologists approached us during two agile conferences (see Sect. 3). We set the names of agile scaling frameworks whose methodologists participated in our survey in bold.

[5] Questionnaire link: https://bit.ly/2ZPl69S.

Table 1. Overview of agile scaling frameworks based on [17]

Framework	Methodologist	Organization	Publ. date	Category	Scaling level
Dynamic Systems Development Method Agile Project Framework for Scrum (DSDM)	Arie van Bennekum	DSDM Consortium	1997	Framework	Portfolio
Crystal Family (Crystal)	Alistair Cockburn		1998	Set of methods	Team
Scrum of Scrums (SoS)	Jeff Sutherland; Ken Schwaber	Scrum Inc	2001	Mechanism	Program
Large Scale Scrum (LeSS)	Craig Larman; Bas Vodde	LeSS Company B.V	2007	Framework	Enterprise
Gill Framework (Gill)	Asif Qumer; Brian Henderson-Sellers	Adapt Inn	2008	Framework	Enterprise
Enterprise Transition Framework (ETF)	–	agile42	2011	Framework	Enterprise
Mega Framework (Mega)	Rafael Maranzato; Marden Neubert; Paula Heculano	Universo Online S.A	2011	Framework	Portfolio
Scaled Agile Framework (SAFe)	Dean Leffingwell	Scaled Agile Inc	2011	Framework	Enterprise
Disciplined Agile Delivery (DAD)	Scott Ambler	Disciplined Agile Consortium	2012	Framework	Enterprise
Enterprise Agile Delivery and Agile Governance Practice (EADAGP)	Erik Marks	AgilePath	2012	Set of practices	Enterprise
Spotify Model (Spotify)	Henrik Kniberg; Anders Ivarsson; Joakim Sundén	Spotify	2012	Model	Enterprise
Recipes for Agile Governance in the Enterprise (RAGE)	Kevin Thompson	Cprime	2013	Framework	Portfolio
Continuous Agile Framework (CAF)	Andy Singleton	Maxos LLC	2014	Framework	Program
Enterprise Scrum (eScrum)	Mike Beedle†	Enterprise Scrum Inc	2014	Framework	Enterprise
eXponential Simple Continuous Autonomous Learning Ecosystem (XSCALE)	Peter Merel	Xscale Alliance	2014	Set of principles	Enterprise
Holistic Software Development (HSD)	Mike MacDonagh; Steve Handy	Holistic Software Consulting Ltd	2014	Framework	Enterprise
ScALeD Agile Lean Development (SALD)	Peter Beck; Markus Gärtner; Christoph Mathis; Stefan Roock; Andreas Schliep	–	2014	Set of principles	Enterprise
FAST Agile (FAST)	Ron Quartel	Cron Technologies	2015	Set of methods	Program
Lean Enterprise Agile Framework (LEAF)	–	LeanPitch Technologies	2015	Framework	Enterprise
Nexus (Nexus)	Ken Schwaber	Scrum.org	2015	Framework	Program
Parallel Agile (Parallel)	Doug Rosenberg; Brarry Boehm; Matt Stephens; Charles Suscheck; Shobha Dhalipathi; Bo Wang	Parallel Agile Inc	2016	Set of methods	Enterprise
Scrum at Scale (S@S)	Jeff Sutherland; Alex Brown	Scrum Inc	2018	Framework	Enterprise

behind the framework creation and the claimed benefits and challenges. The second section presented questions about framework evolution, e.g., the framework version history. In the third section, we aimed to capture the lean and agile foundations behind the framework, e.g., agile practices adopted to develop the framework. In the final section, we collected information on compatibility between the frameworks. The questions were compiled based on previous studies [8,17] and the Ask Matrix[6].

Survey Validation. Two experienced researchers validated the questionnaire from the software engineering research group at TU Munich. Their suggestions on length, language, and the order of questions were incorporated.

Data Collection. We collected data between August 2017 and September 2019 using the online tool *Unipark*[7]. We used various approaches to reach out to the inventors or organizations, i.e., methodologists, that created the frameworks shown in Table 1. First, we sent out the questionnaire link to 22 methodologists by email. Second, we contacted some of the methodologists in two of the leading agile conferences: XP 2019[8] and Agile 2019[9], and emailed them the survey link. Third, we reached a few methodologists via LinkedIn[10] by sending a personal message with the survey link. We received responses from 15 creators.

Data Analysis. We imported the survey data related to our four research questions to excel sheets. The first two authors analyzed data for all research questions individually by following Corbin and Strauss's coding guidelines [21]. We started with breaking down the data into meaningful entities, i.e., open codes. Later, based on the constant comparison of similarities and differences, we grouped the open codes into higher categories of codes called axial codes. Finally, both authors had a few discussions to compare the open and axial codes from their analysis. The majority of the codes matched between the two authors, and only a few adjustments were made by mutual agreement.

4 Results

4.1 RQ1: Evolution of the Agile Scaling Frameworks

Figure 2 shows a time-based overview of the 15 agile scaling frameworks whose methodologists participated in our survey. Grey rectangles ( ) indicate the *start of development* of a framework, whereas green rectangles ( ) show *current versions* and blue rectangles ( ) symbolize *intermediate versions*. Figure 2 also shows two types of dependencies between the frameworks and their versions: Dashed arrows indicate the influence between different frameworks, whereas solid arrows show a predecessor relationship.

[6] http://www.agilescaling.org/ask-matrix.html, last accessed on: 03-10-2021.
[7] https://www.unipark.com/en/, last accessed on: 03-10-2021.
[8] https://www.agilealliance.org/xp2019/, last accessed on: 03-10-2021.
[9] https://www.agilealliance.org/agile2019/, last accessed on: 03-10-2021.
[10] https://www.linkedin.com/, last accessed on: 03-10-2021.

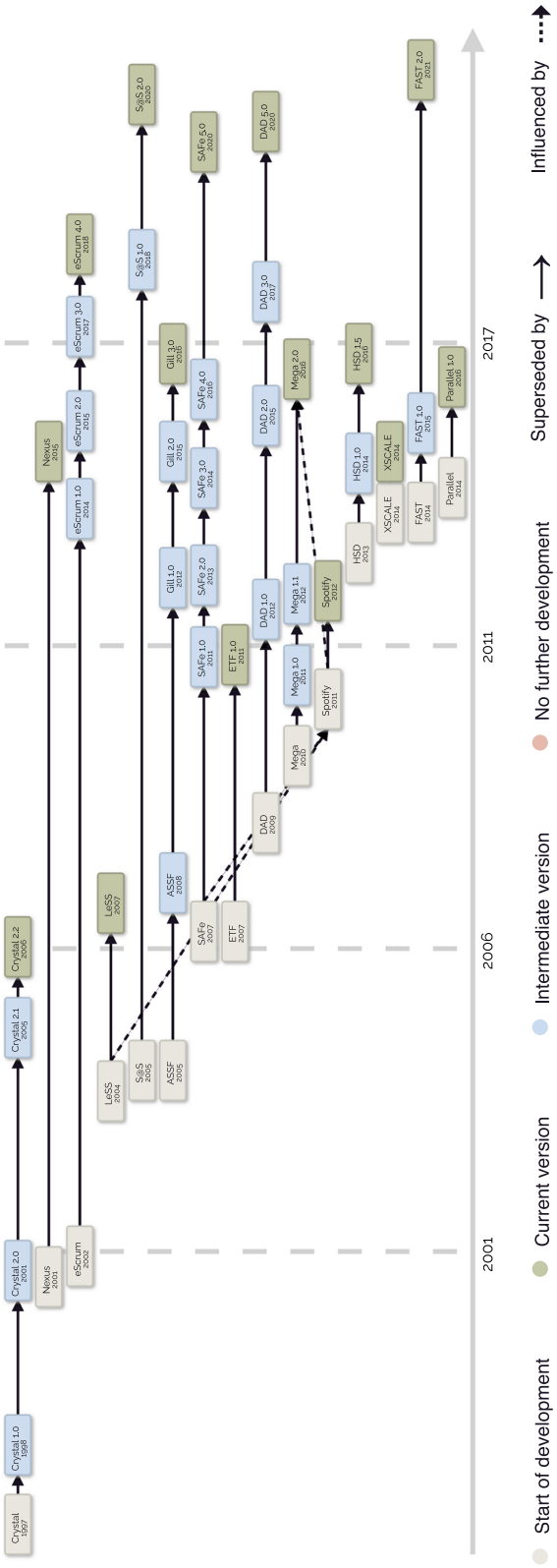


Fig. 2. Evolution of agile scaling frameworks

According to our survey data, *Crystal* is the first created agile scaling framework which development started in 1997. *Nexus*, *eScrum*, and *S@S* were also relatively early designed compared to most other agile scaling frameworks. However, it took the methodologists almost ten years to publish these frameworks, e.g., by publishing their official guides. Although nine frameworks were created before 2010, only three of them went public before 2010. Whereas, between 2011 and 2018, twelve frameworks were published. None of the methodologists indicated stopping the further development of their frameworks. Most frameworks have multiple versions, whereas four frameworks have only one version, namely *Nexus*, *LeSS*, *Spotify*, and *XSCALE*. *Gill* was initially created as the *ASSF* framework (2005-2008), which then evolved into *Gill* in 2012. The methodologists of *Mega* indicated that *Mega 1.0* was a derivative of *SoS*. They also stated that *Mega 2.0* was influenced by *Spotify* including the idea to extend the adoption of agile practices to other parts of the organization. The methodologists of *Spotify* found inspiration from Craig Larman's and Bass Vodde's two books (cf. [22,23]), that later became *LeSS*. *Spotify* was also influenced by the Program Increment Planning events of *SAFe* (cf. [24]).

4.2 RQ2: Key Reasons Behind Creating Agile Scaling Frameworks

Table 2 presents 12 reasons behind creating scaling frameworks based on our survey. These reasons were grouped into four categories: complexity, customer, market, and organization. The most commonly stated reasons were: *improving the agility/adaptability of the organization*, *improving the collaboration of agile teams working on same product*, *improving the coordination of agile teams working*, and *improving the synchronization of agile teams working on same product.*

Table 2. Reasons behind the creation of agile scaling frameworks

Reason category	Reason	Reported in
Complexity	Dealing with increased complexity	ETF, SAFe
	Descaling large product organizations in smaller independent entities	eScrum, XSCALE
Customer	Delivering higher business value	LeSS
	Improving customer involvement	eScrum
Market	Improving the agility/adaptability of the organization	DAD, Gill, HSD, SAFe, S@S, Spotify
	Dealing with changing environments	LeSS
Organization	Improving the collaboration of agile teams working on same product	Nexus, Parallel, SAFe, S@S
	Improving the coordination of agile teams working on same product	Crystal, Nexus, S@S
	Improving the synchronization of agile teams working on same product	FAST, Nexus, SAFe
	Enabling the information/communication flow between agile teams	Crystal, Mega
	Scaling agile to more people/teams/higher organizational levels	LeSS, SAFe
	Managing dependencies between agile teams	eScrum

4.3 RQ3: Benefits of Adopting Agile Scaling Frameworks

Table 3 presents 30 claimed benefits of adopting scaling frameworks based on our survey. These benefits were grouped into two categories, namely: business/product and organization/culture. The most commonly mentioned benefits were: *enabling frequent product deliveries*, *enhancing employee satisfaction/motivation/engagement*, *improving software quality*, *providing customer/business value*, *improving the collaboration of agile teams working on same product*, *improving the coordination of agile teams working on same product*, *improving the synchronization of agile teams working on same product*.

4.4 RQ4: Challenges of Adopting Agile Scaling Frameworks

Table 4 presents 22 challenges of adopting scaling frameworks based on our survey. These challenges were grouped into three categories: implementation, organization/culture, and scope. The most commonly mentioned challenges were: *using frameworks as cooking recipes* and *using frameworks without understanding for what reasons they should be applied*.

Table 3. Claimed benefits of adopting agile scaling frameworks

Benefit category	Benefit	Reported in
Business/Product	Enabling frequent product deliveries	FAST, Parallel, SAFe, S@S, Spotify
	Improving software quality	Mega, Parallel, SAFe
	Providing customer/business value	LeSS, Mega, S@S
	Enabling continuous improvement	ETF, Spotify
	Enabling continuous integration	Mega, Nexus
	Enabling shorter feedback cycles	FAST, Nexus
	Enabling better adaptability to changing market conditions	S@S
	Enabling faster time-to-market	SAFe
	Enabling the release of working products every Sprint	Nexus
	Improving customer satisfaction	eScrum
	Improving efficiency	Gill
	Minimizing software production costs	Parallel
Organization/ Culture	Enhancing employee satisfaction/motivation/engagement	eScrum, FAST, Mega, SAFe, S@S, Spotify
	Fostering the creation of autonomous teams	eScrum, FAST, S@S
	Improving the collaboration of agile teams working on same product	Crystal, Mega, S@S
	Improving the coordination of agile teams working on same product	Crystal, Mega, S@S
	Improving the synchronization of agile teams working on same product	Crystal, Mega, S@S
	Enabling enterprise agility	LeSS, S@S
	Fostering innovation	FAST, Gill
	Improving agile mindset and understanding	DAD, ETF
	Improving accountability	Nexus
	Improving organizational performance	Spotify
	Improving team cohesion	Mega
	Improving transparency	Nexus
	Improving workflows	HSD
	Enabling better understanding of the organization and its vision	DAD
	Enabling the prioritization of company bottlenecks	XSCALE
	Fostering servant leadership	FAST
	Reducing headcount	FAST
	Resolving organizational impediments	S@S

Table 4. Claimed challenges of adopting agile scaling frameworks

Challenge category	Challenge	Reported in
Implementation	Implementing is difficult due to framework complexity	eScrum, HSD
	Missing familiarization with framework	eScrum, Nexus
	Implementation overhead	SAFe
	Misconception due to unconventional agile practices	FAST
	Insufficient guidance	Crystal
	Insufficient guidance regarding lean practices	Mega
Organization/Culture	Using frameworks as cooking recipes	DAD, HSD, SAFe, Spotify
	Using frameworks without understanding for what reasons they should be applied	ETF, Nexus, Spotify
	Lack of management buy-in	SAFe, S@S
	Moving away from agile	Parallel, SAFe
	Moving back from agile to traditional management approaches	ETF, LeSS
	Change resistance	LeSS
	Implementation is difficult in command and control-style organizations	FAST
	Implementation is difficult in traditional organizations	FAST
	Involving non-development units is difficult	Gill
	Implementation is difficult due to remaining power structures	LeSS
	Changing the mindset of the organization is difficult	ETF
Scope	Implementation is limited to team level	Mega
	Implementation is not suitable for monolithic applications	Mega
	Insufficient guidance regarding product backlog management	Mega
	Insufficient guidance regarding managers and specialist positions	LeSS
	Requiring co-location of agile teams	FAST

5 Discussion and Conclusions

5.1 Key Findings

RQ1: How did agile scaling frameworks evolve over the years?
By comparing the evolution map of agile scaling frameworks in Fig. 2 with the evolutionary map of agile methods by Abrahamsson et al. [13], we observed two notable parallels. First, similar to the movement of agile methods, the movement to agile scaling frameworks emerged from parallel innovation both by some inventors of existing agile methods and by consultants who supported organizations in scaling the agile methods. Second, likewise to agile methods, agile scaling frameworks have been continuously emerging and evolving after the movement started. This trend will likely continue as the methodologists of agile scaling frameworks seem to be committed to improving them in the future. Although the evolution map visualizes several agile scaling frameworks, users have concentrated on a few frameworks [25], particularly on *SAFe* and *SoS* [8]. The most recent State of Agile survey [8] confirms this by stating that 35% of their respondents adopted *SAFe* and 16% used *SoS*. A similar observation can be made for agile methods, as 58% of the respondents of the State of Agile survey use Scrum, making it the most commonly used agile method [8].

RQ2: What are key reasons behind creating of agile scaling frameworks?
In total, we found 12 reasons behind the creation of 15 agile scaling frameworks. The reasons identified in our survey fall into either the category of improving the current state of the organization or dealing with the organization's prevalent challenges. Both look similar to reasons that trigger an organizational change [26]. Several reasons, e.g., *improving the collaboration and coordination agile teams working on same product* and *dealing with changing environments* were found in previous studies on large-scale agile development [27,28]. Other reasons related to the scaling of agile methods, such as *dealing with increased complexity* and *scaling agile to more people*, were also reported in [9,29–31]. However, to our knowledge, two reasons found in our survey related to *descaling large product organizations into smaller independent entities* and *improving customer involvement* were not reported by the extant literature on agile development. Surprisingly, several popular reasons for agile, e.g., improving productivity, improving visibility, and improving predictability, were not reported as reasons [8]. As the questionnaire's question was about the main reasons of creating a framework, these earlier mentioned reasons can be some of the implicit reasons behind the creation of the 15 agile scaling frameworks.

RQ3: What are the claimed benefits of adopting agile scaling frameworks?
In total, we identified 30 claimed benefits. The majority of these claimed benefits were similar to the benefits of agile adoption in general found from recent studies on agile method, e.g., State of Agile survey [8]. However, the most common benefit of agile, namely improved productivity [8], was not mentioned by any methodologists. We also identified benefits related to *reducing headcount* and *fostering servant leadership*, which were not found in the previous literature on

large-scale agile development. More research on benefits is needed to establish scientific evidence of using these frameworks in the industry. It is also crucial to understand which practices have contributed to these benefits.

RQ4: What are the challenges of adopting agile scaling frameworks?
We identified 22 challenges from 15 scaling frameworks. To our knowledge, none of the framework's official websites has given information related to the difficulties encountered while adopting these frameworks. The most common challenges identified in our study, i.e., *using frameworks as cooking recipes* and *using frameworks without understanding for what reasons they should be applied*, were not reported by previously published empirical studies. The majority of the challenges found in our study, e.g., *change resistance*, *moving away from agile*, *implementation is difficult due to remaining power structures*, and *lack of management buy-in*, were already reported in previously published studies on scaling frameworks [16,32–34] and large-scale agile development [5,35]. The challenges look similar to agile transformation challenges in general. Hence, using an agile scaling framework is not a silver bullet for scaling agile in large organizations, but a starting point for an agile transformation [33]. Several methodologists mentioned that leaders and change agents should focus on changing people's culture and mindset, rather than using frameworks only as cooking recipes.

5.2 Limitations

We discuss the limitations of our study through the threats, as suggested by Wohlin et al. [36].

Construct Validity. This threat is concerned whether the questions presented in the questionnaire represent the attributes being measured. Two survey experts thoroughly checked the questionnaire and evaluated its' understandability, clarity, and readability to counteract this threat. Moreover, the questions were compiled based on previously published studies in the realm of agile software development.

External Validity. This threat is about the generalizability of the results. We aimed to collect responses from all existing scaling frameworks. Out of 22 frameworks, we received responses from 15 methodologists. We could not get responses from the methodologists of seven frameworks despite contacting them several times via email. Thus, this threat could not be completely mitigated. However, we received responses from the most widely adopted scaling frameworks, such as *SAFe*, *LeSS*, *DAD*, and *Spotify* [8].

Internal Validity. This threat is concerned with factors that can affect the relationship between the research process and survey results, i.e., the cause and effect relationship. We contacted the methodologists via emails found from the frameworks' official websites. We received confirmation from most methodologists after they filled in the survey, which ensured that the right persons answered the survey. We also met some methodologists during the agile conferences personally and asked them to answer the survey.

Conclusion Validity. This threat deals with the ability to conclude from survey data. The data was coded independently by two researchers. Both researchers compared the codes and drew conclusions together to avoid misinterpretation and misunderstanding of the data.

5.3 Conclusions

Large-scale agile development has received significant interest by practitioners and academics over the last years [37]. As organizations are driven by pressures to scale and to react fast, agile scaling frameworks are increasingly prevalent in contemporary software organizations [7,8], sparking a growing academic interest in studying the adoption of these frameworks [6]. Although there is a body of knowledge on agile scaling frameworks, less research has been conducted to provide an overview of these frameworks and their evolution, study the reasons behind creating these frameworks, and investigate the benefits and challenges of adopting these frameworks. We surveyed the methodologists behind the agile scaling frameworks to address this research gap.

Our study provides an overview of 22 agile scaling frameworks of which 15 were covered by our survey. Our study extends extant literature by providing a map on agile scaling frameworks with their evolutionary paths. Although many methodologists started creating their first frameworks between 2001 and 2011, most guides on these frameworks were published later on. Our findings show a cluster of framework publications between 2011 and 2018, confirming the rising industry interest in scaling the agile methods. We identified 12 reasons behind the creation of the agile scaling frameworks. We revealed two new reasons which were not reported by the existing literature on agile development: *descaling large product organizations into smaller independent entities* and *improving customer involvement*. Further, the methodologists claimed 30 different benefits of adopting their frameworks related to business, product, organizational, and cultural aspects. The methodologists also reported two new benefits which were not described in the previous literature: *reducing headcount* and *fostering servant leadership*. The methodologists recognized 22 challenges in the adoption of the frameworks of which two were newly discovered in our study, i.e., *using frameworks as cooking recipes* and *using frameworks without understanding for what reasons they should be applied*.

We encourage researchers to investigate further how contextual factors, such as complexity, multi-product development, or agile maturity, impact a scaling framework's selection. We call for cross-case analyses to compare the adoption of agile scaling frameworks based on common comparison characteristics.

References

1. Dingsøyr, T., Nerur, S., Balijepally, V., Moe, N.B.: A decade of agile methodologies: Towards explaining agile software development. J. Syst. Softw. **85**(6), 1213–1221 (2012). Special Issue: Agile Development

2. Kettunen, P.: Extending software project agility with new product development enterprise agility. Softw. Process Improv. Practice **12**(6), 541–548 (2007)
3. Boehm, B.: Get ready for agile methods, with care. Computer **35**(1), 64–69 (2002)
4. Dingsøyr, T., Moe, N.B., Fægri, T.E., Seim, E.A.: Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation. Empir. Softw. Eng. **23**(1), 490–520 (2017). https://doi.org/10.1007/s10664-017-9524-2
5. Dikert, K., Paasivaara, M., Lassenius, C.: Challenges and success factors for large-scale agile transformations: a systematic literature review. J. Syst. Softw. **119**, 87–108 (2016)
6. Uludag, Ö., Philipp, P., Putta, A., Paasivaara, M., Lassenius, C., Matthes, F.: Revealing the state-of-the-art in large-scale agile development: A systematic mapping study. arXiv preprint arXiv:2007.05578 (2021)
7. Carroll, N., Conboy, K.: Applying normalization process theory to explain large-scale agile transformations. In: Proceedings of the 14th International Research Workshop on IT Project Management, January 2019
8. VersionOne: 14th Annual State of Agile Survey (2020). https://stateofagile.com/#ufh-i-615706098-14th-annual-state-of-agile-report/7027494. Accessed 03 Oct 2021
9. Pries-Heje, J., Krohn, M.M.: The safe way to the agile organization. In: Proceedings of the XP2017 Scientific Workshops, pp. 1–4. ACM, May 2017
10. Alqudah, M., Razali, R.: A review of scaling agile methods in large software development. Int. J. Adv. Sci. Eng. Inf. Technol. **6**(6), 828–837 (2016)
11. Paasivaara, M., Behm, B., Lassenius, C., Hallikainen, M.: Large-scale agile transformation at ericsson: a case study. Empir. Softw. Eng. **23**(5), 2550–2596 (2018)
12. Dingsøyr, T., Fægri, T.E., Itkonen, J.: What is large in large-scale? *A Taxonomy of Scale for Agile Software Development*. In: Jedlitschka, A., Kuvaja, P., Kuhrmann, M., Männistö, T., Münch, J., Raatikainen, M. (eds.) PROFES 2014. LNCS, vol. 8892, pp. 273–276. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13835-0_20
13. Abrahamsson, P., Warsta, J., Siponen, M.T., Ronkainen, J.: New directions on agile methods: a comparative analysis. In: Proceedings of the 25th International Conference on Software Engineering, pp. 244–254. IEEE, May 2003
14. Ambler, S.W.: agile software development at scale. In: Meyer, B., Nawrocki, J.R., Walter, B. (eds.) CEE-SET 2007. LNCS, vol. 5082, pp. 1–12. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85279-7_1
15. Maples, C.: Enterprise agile transformation: the two-year wall. In: Proceedings of the 2009 Agile Conference, pp. 90–95. IEEE, August 2009
16. Conboy, K., Carroll, N.: Implementing large-scale agile frameworks: challenges and recommendations. IEEE Softw. **36**(2), 44–50 (2019)
17. Uludağ, Ö., Kleehaus, M., Xu, X., Matthes, F.: Investigating the role of architects in scaling agile frameworks. In: Proceedings of the 21st International Enterprise Distributed Object Computing Conference, IEEE, pp. 123–132, October 2017
18. Diebold, P., Schmitt, A., Theobald, S.: Scaling agile: how to select the most appropriate framework. In: Proceedings of the 19th International Conference on Agile Software Development: Companion, pp. 1–4. ACM, May 2018
19. Linåker, J., Sulaman, S.M., Maiani de Mello, R., Höst, M.: Guidelines for conducting surveys in software engineering (2015)
20. Punter, T., Ciolkowski, M., Freimut, B., John, I.: Conducting on-line surveys in software engineering. In: International Symposium on Empirical Software Engineering, pp. 80–88. IEEE (2003)

21. Corbin, J.M., Strauss, A.L.: Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory, 3rd edn. Sage Publications Inc., Los Angeles, Calif (2008)
22. Larman, C.: Scaling lean & agile development: thinking and organizational tools for large-scale Scrum. Pearson Education India (2008)
23. Larman, C., Vodde, B.: Practices for scaling lean & Agile development: large, multisite, and offshore product development with large-scale scrum. Pearson Education (2010)
24. Scaled Agile Inc.: PI Planning (2021). https://www.scaledagileframework.com/pi-planning/. Accessed 03 Oct 2021
25. Putta, A., Paasivaara, M., Lassenius, C.: Benefits and challenges of adopting the scaled agile framework (SAFe): preliminary results from a multivocal literature review. In: Kuhrmann, M., et al. (eds.) PROFES 2018. LNCS, vol. 11271, pp. 334–351. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03673-7_24
26. Scaled Agile Inc.: Reasons for SAFe Adoption (2021). https://www.scaledagileframework.com/reaching-the-tipping-point/. Accessed 03 Oct 2021
27. Paasivaara, M.: Adopting safe to scale agile in a globally distributed organization. In: Proceedings of the 12th International Conference on Global Software Engineering, pp. 36–40. IEEE, May 2017
28. Gustavsson, T.: Dynamics of inter-team coordination routines in large-scale agile software development. In: Proceedings of the 27th European Conference on Information Systems, pp. 1–6, June 2019
29. Heikkilä, V.T., Paasivaara, M., Rautiainen, K., Lassenius, C., Toivola, T., Järvinen, J.: Operational release planning in large-scale scrum with multiple stakeholders-a longitudinal case study at f-secure corporation. Inf. Softw. Technol. **57**, 116–140 (2015)
30. McMunn, D., Manketo, P.: Building strong foundations... underwriting fannie mae's agile transformation. In: International Conference on Agile Software Development, Agile Alliance, August 2017
31. Michelson, C., Adolph, S.: Bias from the bottom: A different way to bootup a safe train. In: International Conference on Agile Software Development, Agile Alliance (2019)
32. Putta, A., Paasivaara, M., Lassenius, C.: Benefits and Challenges of Adopting the Scaled Agile Framework (SAFe): preliminary results from a multivocal literature review. In: Kuhrmann, M., et al. (eds.) PROFES 2018. LNCS, vol. 11271, pp. 334–351. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03673-7_24
33. Putta, A., Paasivaara, M., Lassenius, C.: How are agile release trains formed in practice? a case study in a large financial corporation. In: Kruchten, P., Fraser, S., Coallier, F. (eds.) XP 2019. LNBIP, vol. 355, pp. 154–170. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-19034-7_10
34. Kalenda, M., Hyna, P., Rossi, B.: Scaling agile in large organizations: Practices, challenges, and success factors. Journal of Software: Evolution and Process **30**, (2018)
35. Uludağ, Ö., Kleehaus, M., Dreymann, N., Kabelin, C., Matthes, F.: Investigating the adoption and application of large-scale scrum at a German automobile manufacturer. In: Proceedings of the 14th International Conference on Global Software Engineering, pp. 22–29. IEEE, May 2019

36. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer, Cham (2012)
37. Dingsøyr, T., Falessi, D., Power, K.: Agile development at scale: the next frontier. IEEE Softw. **36**(2), 30–38 (2019)

Investigating the Adoption and Application of Large-Scale Scrum at a German Automobile Manufacturer

Ömer Uludağ*, Martin Kleehaus*, Niklas Dreymann*, Christian Kabelin†, Florian Matthes*
*Chair for Informatics 19, Technische Universität München (TUM), D-85748, Garching
Email:{oemer.uludag,martin.kleehaus,niklas.dreymann,matthes}@tum.de
†Ventum Consulting, D-80797, München
Email:{christian.kabelin}@ventum.de

***Abstract*—Over the last two decades, agile methods have been adopted by an increasing number of organizations to improve their software development processes. In contrast to traditional methods, agile methods place more emphasis on flexible processes than on detailed upfront plans and heavy documentation. Since agile methods have proved to be successful at the team level, large organizations are now aiming to scale agile methods to the enterprise level by adopting and applying so-called scaling agile frameworks such as Large-Scale Scrum (LeSS) or Scaled Agile Framework (SAFe). Although there is a growing body of literature on large-scale agile development, literature documenting actual experiences related to scaling agile frameworks is still scarce. This paper aims to fill this gap by providing a case study on the adoption and application of LeSS in four different products of a German automobile manufacturer. Based on seven interviews, we present how the organization adopted and applied LeSS, and discuss related challenges and success factors. The comparison of the products indicates that transparency, training courses and workshops, and change management are crucial for a successful adoption.**

***Index Terms*—large-scale agile development, scaling agile frameworks, large-scale scrum, case study**

I. Introduction

Emerging in the 1990s, agile software development methods such as Extreme Programming [1] and Scrum [2] have transformed and brought unprecedented advancements to software development practice by emphasizing change tolerance, team collaboration, and customer involvement [3], [4]. With these methods, small, co-located, self-organizing teams work closely with the business customer on a single-project context, maximizing customer value and software product quality through rapid iterations and frequent feedback loops [3]. Since agile methods have proved to be successful at the team level, large organizations are now aiming to scale agile methods to the enterprise level [5]. Version One's 12th survey on the state of agile [6] also reflects this industry trend towards adopting agile methods in-the-large. The survey shows that 52% of the 1492 respondents work in companies where the majority of teams are agile. However, the adoption entails new challenges, such as inter-team coordination, dependencies to other existing environments or general resistances to changes [7], [8]. Mainly promoted by consultants, scaling agile frameworks such as the Scaled Agile Framework (SAFe), Disciplined Agile Delivery (DAD), and Large-Scale Scrum (LeSS) [4], [5] pledge to resolve the aforementioned issues. Although the number of organizations using scaling agile frameworks is increasing, scientific literature providing in-depth case studies on the adoption and application is still scarce [6], [7]. We aim to fill this gap by presenting a case study on the adoption and application of LeSS in four different products of a German automobile manufacturer. Based on these objectives, our two research questions are:

- *Research Question 1 (RQ 1): How has LeSS been adopted in different products?*
- *Research Question 2 (RQ 2): How is LeSS applied in different products?*

The remainder of this paper is structured as follows. In Section II, we present the background of our paper and provide an overview of related works. In Section III, we present the research approach of this paper. We briefly describe the case organization and present the results of our case study in Section IV. We discuss the main findings in Section V before concluding the paper with a summary of our results and remarks on future research in Section VI.

II. Background and Related Work

A. Large-Scale Scrum

The LeSS framework (see Figure I) was released in 2008 based on the experiences of Craig Larman and Bas Vodde [9]. It extends Scrum with scaling rules and guidelines without losing sight of Scrum's original goals. Unlike traditional Scrum, LeSS specifies organizational changes. Furthermore, it aims to facilitate coordination between multiple Scrum teams by having a product owner (PO) responsible for a central backlog and several teams. Coordination between teams is done similarly to Scrum where they perform a sprint planning and sprint review. For smaller products, all product members join the same sprint planning and review. For bigger products, a team representative should be sent to the meetings. Although LeSS aims to solely work on principles, it still comprises the following four components [10]:

- **Rules:** Rules define the foundation of LeSS. Similar to Scrum, the focus lies on the structure of teams, roles

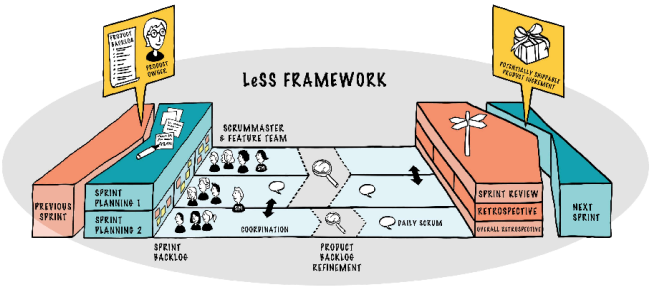


Fig. I: Overview of the LeSS framework [10].

within the team, definition of the requirements of the product, and the development process.

- **Principles:** Principles provide answers on how to apply LeSS in specific enterprise contexts.
- **Guides:** Guides support the adaptation of the rules and a subset of the experiments by providing tips and best practices.
- **Experiments:** LeSS encourages teams to experiment, fail, and learn new concepts.

When organizations aim to scale over eight teams, *LeSS Huge* should be used. LeSS Huge introduces additional elements that are necessary to manage hundreds of people, such as the concept of requirements areas (RAs). RAs are organized around customer-centric requirements. All RAs follow the same sprint cadence and aim for continuous integration across the entire product. An area PO (APO) focuses on one RA and is responsible for an area product backlog (APB). The APO acts essentially the same way as the PO would in the smaller LeSS framework.

B. Related Work

According to Version One's 12th survey on the state of agile, 29% of 1,492 respondents reported using SAFe and 5% LeSS. Although the number of organizations using scaling agile frameworks is increasing, there exists only a handful of papers documenting reported usages [7]. However, some literature reviews have been conducted by academics for studying scaling agile frameworks scientifically. For instance, Alqudah and Razalo [5] provide a literature review on scaling agile frameworks. Therein, they compare seven identified scaling agile frameworks such as DAD, LeSS, Nexus, SAFe, and Spotify based on five criteria: team size, training and certification, adopted methods and practices, required technical practices, and organization type. Based on a structured literature review, Uludağ et al. [11] give a primary analysis of 20 identified scaling agile frameworks such as DAD, Enterprise Scrum, LeSS, Nexus, SAFe, Scrum at Scale, and Spotify. Last but not least, Putta et al. [12] provide a multivocal literature review, which includes both peer-reviewed and non-peer reviewed case studies and experience reports on organizations that have adopted SAFe. Besides these secondary studies, some researchers also present primary studies in the form of case studies on the adoption of scaling agile frameworks. For example, Pries-Heje and Krohn [13] describe a case study from the financial software company SimCorp on how they adopted SAFe. Further, Paasivaara [14] provides a case study highlighting success factors and challenges of the SAFe adoption in the globally distributed software development company Comptel. Moreover, Paasivaara and Lassenius [15], [16] describe a case-study on scaling Scrum in a large globally distributed software development project at the global telecommunications company Nokia. Therein, the authors describe significant challenges the project faced when applying LeSS. Last, Uludağ et al. [17] present a case study from a large insurance company on how they combined domain-driven design with LeSS to support a large-scale agile development program with three agile teams. Although there are some primary and secondary studies, they merely mention LeSS in passing. To the best of our knowledge, literature documenting in-depth case studies on the adoption and application of LeSS in a real-life context is still scarce.

III. Case Study Design

A case study is a suitable research methodology for this paper as we aim to study a contemporary unexplored phenomenon, namely the adoption and application of LeSS, in a complex, real life context [18]. We followed the guidelines described by Runeson and Höst [18] for the case study research process.

Case study design: The main objective of this paper is to study the adoption and application of LeSS in a large IT organization. Based on stated objective, we defined two research questions (see Section I). Our single-case study is of exploratory nature as we analyze an unexplored phenomenon [18]. The case organization was purposefully selected as it provides a unique opportunity to compare different occurrences of LeSS inside a single company and as it represents an information-rich case [19]. Our units of analysis are four products at the car manufacturer's IT department which use LeSS to develop complex software.

Preparation for data collection: We focused on first degree data collection techniques according to Lethbridge et al. [20]. Hence, we collected data by conducting seven semi-structured interviews with one feature team (FT) member, one PO, one agile coach (AC), one scrum master (SM), and three different architects (EA - enterprise architect, SA - solution architect and IA - IT enterprise architect) in four different products of which all adopted and applied the LeSS framework (see Table I). All interviews lasted between 45 minutes to one hour each. The interviews followed a semi-structured questionnaire and were rather conversational in nature to maintain adaptability to the roles and individual experiences of the interviewees. We interviewed seven different roles from four different products to enable triangulation of data sources [18].

Analysis of collected data: All interviews were transcribed and coded using open coding as suggested by Miles et al. [21]. After creating preliminary codes, we refined and consolidated our codes by merging related ones and removing duplicates. Based on that, we looked at groups of code phrases and merged

TABLE I: Overview of interviewed roles and their assigned products.

Role	MPN	OTD	PPM	PFS
PO	-	-	-	1
FT	-	-	-	1
AC / SM	1	1	-	-
EA / SA / IA	-	-	3	-
Total	**1**	**1**	**3**	**2**

them into concepts, which were later related to our formulated research questions.

IV. Results

A. Case Description

The case under investigation concerns a German multinational company that currently manufactures luxury cars and motorcycles and employs more than 120,000 people within the whole organization and around 4500 people in its IT department. In the past, the IT department focused mainly on standardization and cost optimization of the running IT. Driven by digitalization, the IT management saw the increase in agility of the IT department as an essential improvement for its performance and flexibility. At the end of 2016, the IT management decided to transform the IT department into an agile product organization over the next two to three years following the slogan *"100 % agile"*. It committed itself to the goal of transforming all current IT projects to agile and aligning the IT organization completely with agile principles by the end of 2019. Contemporaneously, the autonomous driving department of the case organization chose LeSS as its new working model with the aim of achieving easier communication and coordination, more transparency and shorter decision-making paths throughout the entire department. Success stories of the autonomous driving department with LeSS reached the IT department. In parallel the IT management allowed individual IT projects to choose an appropriate scaling agile framework for themselves, which is why many IT projects decided to adopt LeSS at the beginning of 2017. Still, a large part of software development activities is outsourced to external partners working partly plan-driven.
The four investigated IT projects (from now on products): "Material Part Number" (MPN), "Order-to-Delivery" (OTD), "Product & Price Master Data" (PPM), and "Price Finding Service" (PFS) also decided to adopt LeSS for their product development (see Figure II). MPN aimed to replace the case organization's old legacy system for managing and storing its bills of materials. OTD developed, maintained, and improved IT systems for the case organization's intra-plant logistics. PPM was responsible for the sustainable design of the case organization's master data application. PFS was responsible for the development of a new pricing software.

B. LeSS Adoption

We used exploratory questions for the following categories: (1) timing and duration, (2) reasons and to-be-addressed problems, (3) combined frameworks, (4) training, (5) adaptations, (6) challenges, and (7) lessons learned to evaluate the adoption of the framework.
All four products had recently adopted LeSS for no longer than two years. However, the duration of the adoptions varied between three months to one year because of different complexities. LeSS was introduced to handle inter-team coordination and to balance the limitations of Scrum in large-scale projects. Additional reasons were its moderate degree of complexity while maintaining sufficient guidance for the coordination of multiple agile teams working on the same product. Before the adoption, the products had problems with synchronizing teams, managing their dependencies, and information loss between teams. LeSS was also selected to enable the transition from traditional project thinking to product and customer orientation. During the adoption, LeSS was combined with preexisting lean and agile methods such as Kanban and DevOps. For instance, MPN and PPM adopted LeSS in combination with Kanban to manage epics and features and to track the progress of tasks using Kanban boards. In all products, the adoption was accompanied by a comprehensive training of all employees which range from one single presentation by the SM / AM to a two-days individually adjusted workshop with external experts since employees possess varying levels of prior knowledge in agile software development. The adoption of LeSS in each product led to individual adjustments, e.g., all products extended LeSS by a domain level with a superordinate portfolio layer to coordinate all products within the IT department and align them with the organization's strategic objectives. A so-called "one-calendar" approach, centrally managed by the sub-units of the IT department, eased the adoption by enabling common sprint cadences across products. Although this approach was perceived as contradicting agile values, especially in terms of self-organization, one interviewee stressed its importance:

"*Actually, it's about self-organization in agile. But with a few hundred teams, it's difficult if one has a meeting there or the other one has the daily at other time. Somehow they have to synchronize. Then this one-calendar was our approach to bring structure into it.*"

— SM, OTD

We identified four common problems during the adoption of LeSS. First, although LeSS is minimalist and tries to define roles, artifacts, or processes that are at least needed for large-scale agile development [10], the products had concerns regarding numerous coordination meetings. The SM of OTD described the problem as follows:

"*Many people are simply arrested in thousands of other meetings and if the PO is missing at a meeting such as sprint planning, this is quite sub-optimal.*"

TABLE II: Overview of interviewed products and general information.

	MPN	OTD	PPM	PFS
Start of Adoption	April 2017	Early 2017	End 2017	March 2018
Number of involved Employees	∼600	∼70	∼90	∼40
Sites	Germany	Germany, South Africa	Germany, Poland, Portugal, Russia	Germany, Portugal, Russia
Number of Feature Teams	15	5	6	3
Sprint Lengths (in weeks)	3	3	3	3
LeSS Adoption	LeSS	LeSS	LeSS Huge	LeSS

— SM, OTD

Having too many meetings can reduce the productivity of the employees:

"*The PO attends so many meetings that he doesn't have time to write user stories himself. . . . And if you haven't written them yourself, it's incredibly difficult to accept them.*"

— PO, PFS

Second, due to the currently organizational structure, each product had two POs, one from the specialist department and one from the IT department, which led to the so-called "dual leadership" (see Section IV-C). Third, the employees feared losing status and power. This problem affected primarily middle management employees that were partly retrained to POs during the adoption. Fourth, the adoption was hampered by the lack of agile mindset and understanding about LeSS.
Last but not least, we asked all interviewees about lessons learned. Ex post, clear communication of the big picture of the adoption can increase transparency and sharpen awareness of involved stakeholders. In addition, one FT member mentioned that practical exercises in training courses can ease the adoption:

"*At first glance, it makes the adoption of LeSS easier when practicing and understanding the theory in small projects, preferably in a playful way.*"

— FT, PFS

A summarizing comparison of the four LeSS adoptions can be found in Table III.

C. LeSS Application

We asked the interviewees how their products applied: roles, artifacts, and processes.

Roles: All three standard roles, namely FT, PO, and SM are included in the respective products (see Table IV). In contrast to Scrum, the number of FTs (development teams in Scrum) is two to eight within LeSS. In LeSS Huge, this number can be scaled even higher. Although MPN has 15 FTs, it decided to use multiple LeSS implementations because the subordinate products do not have an overarching product character. FTs of OTD, PPM, and PFS were spread across the world. According to the SM in OTD, one problem was that FTs were still too heavily controlled by external factors and therefore were not entirely self-organized. For PPM, it appeared to be less of a problem. PFS' FTs were already 90 – 95% self-organized. For this reason, there is great potential for improvement by allowing FTs to act as self-organized units. As already indicated in Section IV-B, the role of the PO was shared by several persons, i.e., one PO was responsible for the business side and one for the IT side. This arose from the fact that one person alone did not have the required knowledge to manage the product properly, so the products decided to split the responsibilities. The "dual leadership" created difficulties in coordination for external and internal teams.

"*. . . there still exists one PO on the business and one PO one the IT side but then in the form of a dual leader. If you look at LeSS or Scrum then I have to say having a dual leader is the worst thing to do.*"

— EA, PPM

The suggested improvement was to remove dual leadership by bundling responsibilities into one PO. The EA of PPM suggested that the IT PO should work more with FTs and act as a substituting PO where he could collect valuable insights. In addition, PPM had six product owners who were responsible for subareas, representing APOs as suggested by LeSS Huge [10]. Last but not least, all products introduced the role of the Domain PO (DPO) with more traditional project management functions. Their responsibility included synchronization of the FTs and planning the budget and capacities. They also took overall responsibility for the products and coordinated at portfolio level with each other.

"*They [DPOs] are actually only responsible for the budget and capacities and should not be involved in the actual work.*"

— EA, PPM

Figure II provides an overview of the different PO roles in the respective products. In OTD, the role of the SM did not exist per se, but instead the role of the agile master (AM). The AM of OTD was not only responsible for helping FTs

TABLE III: Comparison of the LeSS adoptions in the four products.

	MPN	OTD	PPM	PFS
Reasons for Adoption	• Simplicity of LeSS • Optimizing product cut • Dealing with high project complexity • Enabling change from project to product	• System optimization goals from LeSS • Success stories of the autonomous driving department • Three-day introduction to LeSS by Craig Larman • Enabling change from project to product	• Dealing with high project complexity • Enabling change from project to product	• Reducing dependencies between agile teams • Minimizing loss of information between agile teams • Handling inter-team coordination
Combined Frameworks	• Combined LeSS with SAFe to have a superordinate portfolio level • Combined LeSS with Kanban to manage epics and features	• Combined LeSS with DevOps so that FTs have the required skill-set and the full responsibility to release software	• Combined LeSS Huge with Kanban to track and monitor the progress of tasks	–
LeSS Training	• 2-day training program for each employee extended with LeSS	• SM coaching • External agile coaches train internal employees who in-turn coach FTs	• 2-day training program for each employee extended with LeSS Huge • External agile coaches train internal employees	• External agile coaches train internal employees
Adaptations	• LeSS extended by a domain level • Common sprint cadences are facilitated by a one-calendar approach	• LeSS extended by a domain level • Common sprint cadences are facilitated by a one-calendar approach	• LeSS Huge extended by a domain level	• LeSS extended by a domain level
Challenges	• Sharing a common vision • Fear of losing power • Communication gaps between products • High complexity of the product • Correct product cuts • Balancing up-front planning vs. emergent design • Dual leadership of the PO role	• Numerous coordination meetings • High complexity due to the number of FTs • Low prior knowledge of agile methods • Traditional line responsibilities of employees complicate their focus on agile working • Dual leadership of the PO role	• Numerous coordination meetings • Splitting large requirements into smaller requirements • APOs keeping own APBs • Synchronizing APBs • Dealing with cultural differences between FTs • Missing transparency regarding roles and responsibilities • Dual leadership of the PO role	• Numerous coordination meetings • Fear of losing power • Specialist departments have low knowledge of LeSS • Establishing agile mindset • Changing roles and responsibilities due to agile working • Dual leadership of the PO role
Lessons Learned	• Communicating the big picture of the adoption to obtain the commitment of all stakeholders • Performing Inspect and Adapt at regular intervals	• Performing Inspect and Adapt at regular intervals	• Reducing responsibilities to a few people to increase decision-making speed	• Practical exercises in training courses deepen the understanding of LeSS

TABLE IV: Identified results for the application of roles within the products.

Role	MPN	OTD	PPM	PFS
FT	✓	✓	✓	✓
PO	✓	✓	✓	✓
SM	✓	✓	✓	✓
Additional Roles	✓	✓	✓	✓

to apply LeSS and self-organize, but also for the change management and people management:

"*The agile master is responsible for the methodology, i.e., method training, shielding the team from stakeholders who want something, eliminating impediments, and organizing events.*"

— SM, OTD

All products involved additional roles which go beyond LeSS. For instance, PFS included the role of the PO support, who was responsible for creating user stories, as the PO had no time for this. PFS also introduced the business analyst (BA) role, who was responsible for dealing with the problems and requirements of FTs. He was also the first point of contact for external parties. Each product was

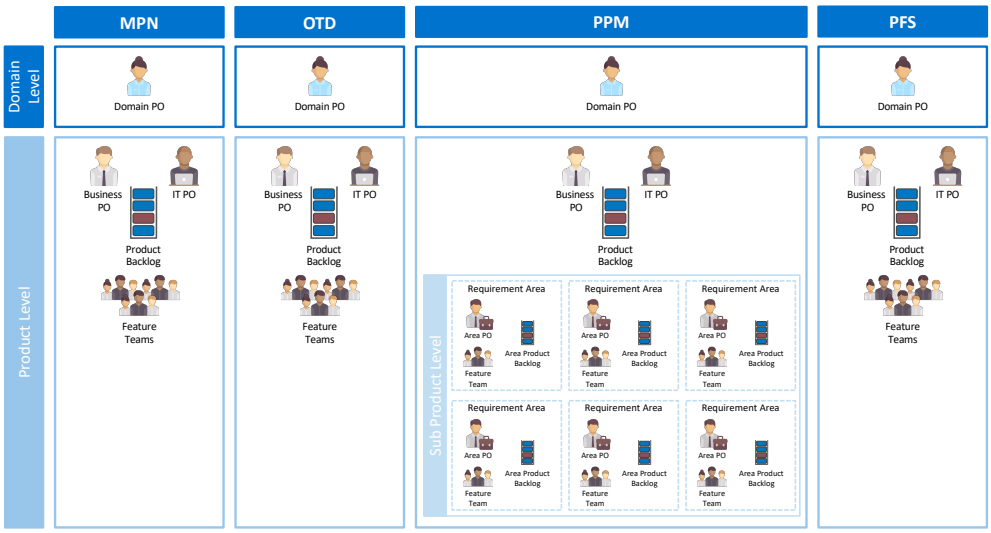


Fig. II: Overview of the different PO roles in the four products.

accompanied by different types of architects, such as IT architects, enterprise architects, and solution architects that mainly acted as external consultants for various architectural topics. Such topics include: showing the positioning of the product in the overall organization context, guiding FTs in the realization of the to-be architecture of the product, and ensuring that FTs adhere to architectural standards. In this context, the EA of PPM also stated that:

"*My personal opinion is that when somebody does not know how to continue they call the architects. They are really just ad-hoc fire extinguishers, who have to come quickly.*"

— EA, PPM

Artifacts: Table V provides an overview of used artifacts. In general, there existed a product backlog for all products. The main difference was that MPN, OTD, and PFS had one product backlog, whereas PPM had one common and six RA-specific product backlogs, so-called area product backlogs. According to one interviewee, this caused the following problem: if there is only one common product backlog on product level for several sub areas, the sole prioritization based on importance becomes difficult, as it cannot generally be said that one sub product is more important than another one. In general, all products had a sprint backlog, sprint goal, and product increments as artifacts. The definition of done (DoD) was implemented by all products and has been discussed at a great deal as they were not fully used. The PO of PFS described the DoD as particularly important, since it defines clear expectations regarding user stories towards external partners. However, PFS was the only product that had a finished DoD. Instead, the interviewee in MPN stated:

"*Basic artifacts as the DoD do exist in every product, however are not used.*"

TABLE V: Identified results for the application of artifacts within the products.

Artifacts	MPN	OTD	PPM	PFS
Product Backlog	✓	✓	✓	✓
Sprint Backlog	✓	✓	✓	✓
Sprint Goal	✓	✓	✓	✓
Product Increment	✗	✓	✓	✗
Definition of Done	✓	✓	✓	✓
Additional Artifacts	✓	✗	✓	✓

— AC, MPN

This was validated by the IA of PPM:

"*Probably it [the DoD] was introduced once in the beginning and now it is in some corner, somewhere in Confluence on some subordinated page, which nobody has accessed for a long time. This sounds very plausible to me.*"

— IA, PPM

The respondents considered the lack of use of the DoD to be problematic, as the product increment was not assessed according to predefined criteria. OTD followed a different approach. An external team developed a general DoD, and then the PO adapted it for his respective features to fit individual requirements. Some interviewees mentioned two additional artifacts. PPM and PFS also used the definition of entry (DoE) which was created by the PO. The DoE was used to describe rough requirements for individual user stories. In addition, MPN, PPM, and PFS used the definition of ready (DoR) as the last step prior to implementation. The DoR was

utilized to describe user stories ready for implementation. However, one interviewee (SA) in PPM stated that the DoE eventually vanished.

TABLE VI: Identified results for the application of processes within the products.

Processes	MPN	OTD	PPM	PFS
Sprint	✓	✓	✓	✓
Product Backlog Refinement	✓	✗	✓	✓
Sprint Planning (1 & 2)	✓	✓	✓	✓
Daily Scrum	✓	✓	✓	✓
Sprint Review	✓	✓	✓	✓
Retrospective (Team & Overall)	✓	✓	✓	✗
Additional Processes	✓	✓	✓	✓

Processes: In general, all LeSS events were widely adopted in all products (see Table VI). According to the SA of PPM, it was difficult to get all relevant people to attend meetings. Meetings were planned in the products at the same time (one-calendar approach), which made cooperation and coordination between people in different meetings very difficult. In addition, retrospectives within PFS were not implemented yet:

"*Well, we haven't been following the retro so closely so far, but we have decided to do more.*"
— FT, PFS

All products organized communities of practices (CoPs) for collaboration and information exchange within technical and business domains across products. One interviewee also added that:

"*Communities of practices are very helpful for the coordination of processes, methods, and tools as well as for comprehensive harmonization and standardization.*"
— SM, OTD

The architecture CoP was the most outstanding CoP within all products as they were implemented by all products. There, architects and other stakeholders discuss architecture-related topics, make decisions, and design architecture guidelines which also affect FTs:

"*There is an architecture community that not only discusses, but can also make decisions. . . . You have to be able to provide input to the teams, but ideally through a community approach and not through strong external roles.*"
— AC, MPN

Additional CoPs were organized for POs and SMs as well as for the testing domain. The former enables DPOs and POs to coordinate and to find a common direction on enterprise level. At regular intervals, MPN and OTD also have organized inspect and adapt events to check where they are in the adoption process, what to improve, and how to adjust their behavior respectively. Last but not least, PPM and PPM organized big events that took place once or twice a year in one of the sites to facilitate team coherence and trust.

V. Discussion

A. Key findings

We now discuss the main outcomes of our findings.

Key findings RQ1: After analyzing different adoptions of LeSS in four products, we identified following five success factors. First, the adoption must be 100% transparent as higher transparency incentivizes FTs to deliver software in higher quality.

Second, comprehensive training courses and workshops ease the adoption since they provide a shared understanding of new practices, roles, and responsibilities. Third, employees should be involved as early as possible in order to minimize change resistance. Fourth, managers should be aware of dissatisfied employees to retain their status and power. Accordingly, managers should raise awareness regarding the value of change to the organization. Fifth, the motivation behind LeSS should be properly promoted, advertised, and communicated.

Key findings RQ2: Based on the four LeSS applications, the following six key findings emerge. First, although the self-organization of FTs were acknowledged within the organization, sprint cadences were centrally organized using a so-called "one-calendar" approach. Second, all four products were extended by a domain level and supported by a DPO as the products were not necessarily independent from each other. Third, due to the present organization structure, the role of the PO was shared by two or three people resulting in the so-called "dual or even triple leadership". Many interviewees complained about this situation since it slowed down processes and hampered what LeSS wants to achieve: fast and agile decisions. Fourth, all products extended LeSS by involving additional roles such as BAs, PO support, shared services, and solution and enterprise architects. Fifth, all four products extended LeSS with additional processes to facilitate the exchange of shared knowledge between the products. These events included various types of communities of practices such as architecture, SM & PO or testing communities. Sixth, the interviewed products organized large team-building events that took place once or twice a year in one of the sites. With those, they aimed at building trust between FTs and overcoming cultural barriers.

B. Threats to validity

We discuss potential threats to validity along with an assessment scheme as suggested by Runeson and Höst [18]. First, we address **construct validity** by interviewing multiple roles across four different products. We also transcribed, coded, and analyzed the interviews. Second, **internal validity** is not relevant, as this research was neither explanatory nor causal

[18]. Third, we address **external validity** by providing a detailed description of the case and focusing on analytical generalization [18]. This paper provides empirical insights that allow for a profound understanding of the adoption and application of LeSS. The presented findings should be viewed as valuable insights for other organizations that aim to adopt and apply LeSS. Last, we ensure the **reliability** of our results by using a case study protocol with detailed procedures for data collection and analysis. We also collected data from different sources by multiple interviewers to allow data and observer triangulation [18].

VI. Conclusion and Future Work

The success of agile methods for small teams inspired large organization to apply them at large-scale by using scaling agile frameworks [4], [5]. Although the number of organizations using these frameworks is increasing, scientific literature providing in-depth analysis is still scarce [6], [7]. We aimed to fill this gap by providing a case study regarding the adoption and application of LeSS in four different products of a German automobile manufacturer. Our findings indicate that a transparent adoption incentivizes teams to deliver high-quality software and that comprehensive training courses and workshops ease the adoption. We also found out that the case organization extended LeSS with additional roles and processes to facilitate the exchange of shared knowledge between products, to build trust between teams, and to adapt it to the current organizational structure.

While we are confident that our findings will contribute to the existing body of knowledge on actual experiences related to LeSS, we encourage other researchers to conduct further in-depth case studies on LeSS and other scaling agile frameworks. For instance, it would be interesting to study to what extent companies have to adapt their organizational structures and processes in order to use LeSS. In future studies, researchers should also conduct cross-case analyses with the goal to compare the adoption and application of LeSS in different types of organizations, e.g., digital natives vs. traditional companies.

References

[1] K. Beck, *Extreme programming explained: embrace change*. Addison-Wesley, 2000.

[2] K. Schwaber and J. Sutherland, "The scrum guide," Scrum.org, Tech. Rep., 2017.

[3] P. Kettunen, "Extending software project agility with new product development enterprise agility," *Software Process: Improvement and Practice*, vol. 12, no. 6, pp. 541–548, 2007.

[4] T. Dingsøyr and N. B. Moe, "Towards principles of large-scale agile development," in *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation*, T. Dingsøyr, N. B. Moe, R. Tonelli, S. Counsell, C. Gencel, and K. Petersen, Eds. Springer, 2014, pp. 1–8.

[5] M. Alqudah and R. Razali, "A review of scaling agile methods in large software development," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 6, no. 6, pp. 828–837, 2016.

[6] VersionOne, "12th annual state of agile report," VersionOne, Tech. Rep., 2018.

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

[8] Ö. Uludağ, M. Kleehaus, C. Caprano, and F. Matthes, "Identifying and structuring challenges in large-scale agile development based on a structured literature review," in *22nd International Conference on Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 2018, pp. 191–197.

[9] C. Larman and B. Vodde, *Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum*. Addison-Wesley, 2008.

[10] ——, *Large-Scale Scrum: More with LeSS*. Addison-Wesley, 2016.

[11] Ö. Uludağ, M. Kleehaus, X. Xu, and F. Matthes, "Investigating the role of architects in scaling agile frameworks," in *21st International Conference on Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 2017, pp. 123–132.

[12] A. Putta, M. Paasivaara, and C. Lassenius, "Benefits and challenges of adopting the scaled agile framework (safe): Preliminary results from a multivocal literature review," in *International Conference on Product-Focused Software Process Improvement*. Springer, 2018, pp. 334–351.

[13] J. Pries-Heje and M. M. Krohn, "The safe way to the agile organization," in *Proceedings of the XP2017 Scientific Workshops*. ACM, 2017, p. 18.

[14] M. Paasivaara, "Adopting safe to scale agile in a globally distributed organization," in *IEEE 12th International Conference on Global Software Engineering*. IEEE, 2017, pp. 36–40.

[15] M. Paasivaara and C. Lassenius, "Scaling scrum in a large distributed project," in *International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2011, pp. 363–367.

[16] ——, "Scaling scrum in a large globally distributed organization: A case study," in *11th International Conference on Global Software Engineering (ICGSE)*. IEEE, 2016, pp. 74–83.

[17] Ö. Uludağ, M. Hauder, M. Kleehaus, C. Schimpfle, and F. Matthes, "Supporting large-scale agile development with domain-driven design," in *International Conference on Agile Software Development*. Springer, 2018, pp. 232–247.

[18] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, p. 131, 2009.

[19] M. Q. Patton, *Qualitative evaluation and research methods*. SAGE Publications, 1990.

[20] T. C. Lethbridge, S. E. Sim, and J. Singer, "Studying software engineers: Data collection techniques for software field studies," *Empirical Software Engineering*, vol. 10, no. 3, pp. 311–341, 2005.

[21] M. B. Miles, A. M. Huberman, and J. Saldana, *Qualitative Data Analysis: A Methods Sourcebook*. SAGE Publications, 2014.

Identifying and Structuring Challenges in Large-Scale Agile Development based on a Structured Literature Review

Ömer Uludağ, Martin Kleehaus, Christoph Caprano, Florian Matthes
Chair for Informatics 19
Technische Universität München (TUM)
D-85748, Garching
{oemer.uludag,martin.kleehaus,christoph.caprano,matthes}@tum.de

***Abstract*—Over the last two decades, agile methods have transformed and brought unique changes to software development practice by strongly emphasizing team collaboration, customer involvement, and change tolerance. The success of agile methods for small, co-located teams has inspired organizations to increasingly apply agile practices to large-scale efforts. Since these methods are originally designed for small teams, unprecedented challenges occur when introducing them at larger scale, such as inter-team coordination and communication, dependencies with other organizational units or general resistances to changes. Compared to the rich body of agile software development literature describing typical challenges, recurring challenges of stakeholders and initiatives in large-scale agile development has not yet been studied through secondary studies sufficiently. With this paper, we aim to fill this gap by presenting a structured literature review on challenges in large-scale agile development. We identified 79 challenges grouped into eleven categories.**

I. Introduction

Emerging in the 1990s, agile software development methods, such as Extreme Programming (XP), Feature-Driven Development, and Scrum, have transformed and brought unprecedented changes to software development practice by strongly emphasizing change tolerance, continuous delivery, and customer involvement [1], [2]. Many enterprises are already using agile methods to maximize customer value and quality of delivered software products, but are uncertain how to introduce them at scale, since they are originally designed for small, co-located teams [1], [3]. This problem is exacerbated by the fact that the adoption of agile methods at larger scale entails new challenges, such as inter-team coordination and communication, dependencies with other organizational units or general resistances to changes [3], [4]. Despite these known challenges, there is an industry trend towards adopting agile methods in-the-large [3], [5].

Compared to the rich body of agile software development literature describing typical challenges (cf. [6], [7] or [8]), challenges in large-scale agile development has not yet been studied through secondary studies sufficiently [3]. Dikert et al. [3] made a first attempt to solve this problem by presenting a systematic literature review of large-scale agile transformations. They identified 35 reported challenges and 29 success factors for large-scale agile transformations. However, the presented challenges are not directly related stakeholders in order to provide appropriate proven solutions for addressing them. In our larger study, we aim to fill this gap by introducing the concept of large-scale agile development patterns and to provide best practices for recurring challenges of stakeholders and initiatives in large-scale agile development. Our study is inspired by the pattern-based approach to Enterprise Architecture Management (EAM) [9]. As a starting point of our study, we present our qualitative findings on stakeholder- and initiative-related challenges in large-scale agile development with the help of a structured literature review. Based on this objective, three research questions (RQ) were formulated

- *RQ1: Which stakeholders exist in large-scale agile development?*
- *RQ2: What are challenges of stakeholders and initiatives in large-scale agile development?*
- *RQ3: What are generalizable findings on stakeholder- and initiative-related challenges in large-scale agile development?*

A. Research methodology

The goal of the literature review is to identify challenges of stakeholders and initiative in large-scale agile development. In order to achieve this goal and to ensure the rigor and relevance of the research, we applied a structured literature review approach as recommended by Brocke et al. [10] that consists of four phases. In the first phase, we defined the review scope and formulated adequate research questions about challenges in large-scale agile development. In the second phase, we identified key concepts by concept mapping, which also provided us the opportunity to obtain relevant search terms: *Agile and Lean Software Engineering*, *Large-Scale Agile Development*, *Agile Transformation*, and *Challenges, challenges and Problems*. These search terms were used in the subsequent literature search in the third phase. We examined a range of different Information Systems journals, conference proceedings using ACM Digital Library, IEEExplore, Scopus, and Web of Science. Having compiled the aforementioned list of search terms, we then used them in electronic full-text search queries. Initially, 67 sources were identified as

relevant, given their focus on the topic, after analyzing a total of 560 sources (title, abstract, and outline). Additionally, we conducted a backward search, resulting in additional 6 sources. In total, we obtained 73 relevant sources. In the fourth phase, we coded the primary studies using a deductive approach as proposed by Cruzes and Dybå [11]. We established an a priori list of codes inspired by the EAM pattern language elements [9], which includes stakeholders, challenges, methodology patterns, architecture principles, viewpoint patterns, and anti-patterns. In the initial step, we started with the actual coding of the data structured by the aforementioned code categories. During the coding, we also identified the relationships between the codes of the different code families. Particularly, we related challenges to respective stakeholders and solutions, such as methodology patterns or architecture principles. For instance, we can determine typical challenges of solution architects and how they are trying to address them based on this structure. After creating preliminary codes, we refined and consolidated our codes by merging related ones and removing duplicates. In the final step, we grouped related challenges into eleven categories: *culture & mindset*, *communication & coordination*, *enterprise architecture*, *geographical distribution*, *knowledge management*, *methodology*, *project management*, *quality assurance*, *requirements engineering*, *Software Architecture*, and *tooling*. Table I presents a description of the codes families and the final state of the coding. Note that in this paper, we only discuss the results related to challenges and stakeholders, and that it as such forms a part of a larger study. In our larger study, we aim to introduce the concept of large-scale agile development patterns, which builds on and extends the idea of the proven pattern-based approach to EAM [9]. The aim of this new pattern language is to address typical problems of stakeholders and initiatives in large-scale agile development.

The remainder of this paper is structured as follows. In Section II, we provide an overview of related works describing challenges and success stories in large-scale agile development. In Section III, we present our findings on large-scale agile development challenges identified in the literature. We discuss the main findings in Section IV before concluding the paper with a summary of our results and remarks on future research in Section V.

II. Related work

Dikert et al. [3] conducted a systematic literature review of industrial large-scale agile transformations focusing on reported challenges and success factors in the transformation. 47 out of 117 relevant papers were selected to obtain 35 challenges and 29 success factors for agile adoption. They grouped the challenges in nine categories and the success factors in eleven categories. Paasivaara and Lassenius [12] validated and deepened these findings with a pilot study. The result is an improved and weighted version of the success factors and challenges. However, there is no relationship to stakeholders that are affected by these challenges. Viswanath [13] observed more than 400 employees in a company during their five years long lean transformation. The employees were analyzed while facing challenges, pitfalls and success factors according to three dimensions: Process, Product and People. Viswanath [13] showed that every stakeholder in a company is involved in the lean transformation. Nevertheless, detailed relations of challenges to stakeholders is also not existing in this paper. Bjarnason et al. [14] reported that agile transformations does not affect only software developers but also other disciplines like requirements engineering. The challenges of over-scoping and communication gaps can be addressed with agile approaches. Some of the to be faced challenges are similar to the traditional way but the transformation cause also new challenges like finding the right balance between agility and stability. Bjarnason et al. [14] also presented that requirements engineering is affected by the agile transformation and have to face new challenges. Unfortunately, there is right now no collection of stakeholder specific challenges that directly mention stakeholders like requirement engineers.

III. Challenges in large-scale agile development

A. Stakeholders in large-scale agile development

In our structured literature review, we identified 40 stakeholder roles that are involved in large-scale agile development. Many of them are either already present in traditional software development or are used as a synonym for another role. We consolidated the 40 roles to 14 stakeholder roles. One example of this consolidation is the role of the Chief Architect [15] which has been merged with the Enterprise Architect because of their similar areas of responsibilities. The following 14 stakeholders were obtained in large-scale agile development:

- Development Team (47)[1],
- Product Owner (33),
- Scrum Master (30),
- Software Architect (21),
- Test Team (18),
- Product Manager (13),
- Program Manager (4),
- Agile Coach (4),
- Enterprise Architect (3),
- Business Analyst (3),
- Solution Architect (2),
- Portfolio Manager (2),
- Support Engineer (2), and
- UX Expert (1).

B. Challenges in large-scale agile development

In total, we identified 79 challenges of which 41 newly arose by large-scale agile development and 38 are strengthened by large-scale agile development (see Table II and Table III).

IV. Discussion

A. Key findings

Let us now reflect on the three research questions described in Section I. ***RQ1: Which stakeholders exist in large-scale***

[1]The number in parentheses stands for the number of documents that mention the respective role.

TABLE I
OVERVIEW OF CODE FAMILIES AND CODES

Code family	Description	Examples	# Identified elements	Codes
Stakeholders	A person with a challenge in a large-scale agile development	Product Owner, Scrum Master, Software Architect	14	770
Challenges	Stakeholders or initiatives are confronted with challenges that have to be addressed in large-scale agile development	Ensuring that non-functional requirements are considered by the Development Team	79	286
M-Patterns	Methodology patterns (M-Patterns) are defined as concrete steps that are performed to address recurring challenges in large-scale agile development	Scrum of scrums, community of practices, creating an architectural runway	88	196
Architecture Principles	Architecture principles define the underlying general rules and guidelines for the use and deployment of all IT resources and assets across the enterprise	Loose coupling of systems or services, reuse of functionalities, buy before make	4	5
V-Patterns	Viewpoint patterns (V-Patterns) are defined as documentations of proven practices to recurring problems for specific contexts in form of viewpoints for the creation of views	Burndown chart, context map, pulse chart	9	12
Anti-Patterns	Anti-patterns detail on typical mistakes in large-scale agile development, and present revised solutions, which help pattern users to prevent these pitfalls	Do not put individual goals over team goals, do not adopt all agile practices in one go, do not overshoot coordination meetings	17	68
		Total		1378

agile development? In the literature review, we observed 40 different stakeholder roles in large-scale agile development. We consolidated them to 14 final stakeholder roles. The stakeholder roles include roles from agile software development as well as new roles, like Software Architects or portfolio managers.

RQ2: What are challenges of stakeholders and initiatives in large-scale agile development? We identified 79 challenges for large-scale agile development (see Table II and Table III). These challenges can be either program-specific or related to specific stakeholders. Thereby, we assigned the challenges either to the in RQ1 observed stakeholder roles or marked them as program-specific.

RQ3: What are generalizable findings on stakeholder- and program-related challenges in large-scale agile development? *Architecture becomes more important the more complex the task or system is.* Although, Software Architects are not included in many agile methods, such as XP or Scrum, they face several challenges in large-scale agile development. Furthermore, more than 20% of the challenges are related to architecture-related topics. All of these challenges newly arose with large-scale agile development. *New stakeholder roles are involved when scaling agile development.* Although, the role of the architect was not intended in agile software development, because it only contains the role of a Product Owner, Scrum Master and the Development Team and additional ones are not mentioned [65]. Nevertheless, we observed in the literature analysis further roles like software and enterprise architects or product managers. *Scaling agile development entails new communication and coordination challenges.* Additional stakeholder roles help to manage big software programs. This includes also the management of multiple agile teams. In the literature review, we identified eight communication and coordination challenges and 75% of them newly arose from large-scale agile development. *Challenges in agile development may still exist in large-scale agile development.* 38 of 79 identified challenges still exist in large-scale agile development. These challenges are typical for agile development and are reinforced by large-scale agile development. *Stakeholders that are successfully isolated by the Scrum Master from external influences have less challenges in large-scaled agile development.* Only 7% of the observed challenges are either challenges for the Development or Test Team. Furthermore, the top challenge topics are inter-team coordination and communication problems as well as architecture related issues. Stakeholders that are isolated by the Scrum Master from external influences are not affected by these challenges and face less challenges in large-scale agile development.

B. Limitations

This paper has a few limitations, which should be mentioned at this point: First, although, we spent much time and effort into developing a suitable search string and conducted a structured database search, there is still a certain chance that not all important contributions have been identified. We found additional literature through a backward search of the analyzed papers in the literature search process. Some relevant studies might have evaded our attention in spite of our best efforts. Second, the initial coding procedure was conducted by only one researcher, which might have led to biased classifications. It might have been better if two researchers had been involved working on a pair coding mode from the beginning.

TABLE II
LARGE-SCALE AGILE DEVELOPMENT CHALLENGES

Name	Challenge Category	Novelty	Relationship to Stakeholders	Number	Sources
Coordinating multiple agile teams that work on the same product	Communication & Coordination	yes	Program Manager	15	[3], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29]
Considering integration issues and dependencies with other subsystems and teams	Software-Architecture	yes	Solution Architect	14	[3], [12], [18], [20], [23], [24], [26], [28], [30], [31], [32], [33], [34], [35]
Coordinating geographically distributed agile teams	Geographical Distribution	yes	Scrum Master	8	[3], [12], [19], [25], [29], [33], [36], [37]
Facilitating shared context and knowledge	Knowledge Management	no	Initiative-Related	7	[13], [21], [31], [33], [38], [39], [40]
Managing technical debts	Software-Architecture	no	Software Architect	7	[13], [28], [29], [33], [41], [42]
Dealing with incorrect practices of agile development	Methodology	no	Agile Coach	7	[12], [21], [25], [32], [33], [41], [43],
Dealing with doubts in people about changes	Culture & Mindset	no	Agile Coach	7	[13], [21], [31], [33], [38], [39], [40]
Ensuring that non-functional requirements are considered by the Development Team	Software-Architecture	yes	Software Architect, Solution Architect	6	[3], [12], [26], [30], [33], [44]
Finding the right balance between architectural improvements and business value	Software-Architecture	yes	Software Architect	6	[26], [28], [30], [31], [33], [45]
Creating precise requirement specifications for the Development Team	Requirements Engineering	no	Product Owner	5	[3], [12], [37], [44], [46]
Obtaining management buy-in	Culture & Mindset	no	Initiative-Related	5	[3], [12], [42], [47], [48]
Providing sufficient tools and infrastructure for remote communications	Tooling	no	Initiative-Related	5	[27], [36], [39], [49], [50]
Sharing common vision	Knowledge Management	yes	Program Manager, Product Owner	5	[3], [13], [31], [51], [52]
Creating a proper upfront architecture design of the system	Software-Architecture	yes	Software Architect	5	[28], [30], [32], [47], [53]
Eliciting and refining requirements of end users	Requirements Engineering	no	Product Owner	5	[3], [12], [25], [26], [37]
Establishing self-organization	Communication & Coordination	no	Development Team	5	[3], [21], [29], [33], [54]
Splitting large and complex requirements into smaller requirements	Requirements Engineering	yes	Product Owner, Program Manager	5	[3], [12], [13], [33], [55]
Dealing with internal silos	Knowledge Management	yes	Initiative-Related	5	[3], [12], [25], [28], [56]
Dealing with increasing workload of key stakeholders	Project Management	yes	Initiative-Related	4	[3], [31], [33], [34]
Facilitating communication between agile teams and other teams using traditional practices	Communication & Coordination	yes	Product Owner	4	[3], [37], [42], [48]
Managing dependencies to other existing environments	Enterprise Architecture	yes	Enterprise Architect	4	[3], [26], [37], [42]
Balancing short-term and long-term goals	Requirements Engineering	no	Product Manager	4	[3], [12], [25], [57]
Establishing a common scope for different stakeholder groups	Knowledge Management	yes	Initiative-Related	4	[29], [34], [39], [49]
Creating team spirit and trust among agile teams	Culture & Mindset	yes	Initiative-Related	4	[19], [29], [33], [41]
Managing and integrating heterogenous subsystems of different Development Teams	Software-Architecture	yes	Solution Architect	3	[3], [28], [44]
Aligning and communicating architectural decisions	Software-Architecture	yes	System Architect, Solution Architect	3	[3], [28], [58]
Managing and sharing knowledge about system components and their dependencies with stakeholders	Enterprise Architecture	yes	Enterprise Architect	3	[3], [21], [28]
Communicating business requirements to Development Teams	Requirements Engineering	no	Product Owner	3	[25], [26], [59]
Facilitating agile teams to participate at cross-shore meetings	Geographical Distribution	yes	Scrum Master	3	[3], [39], [50]
Synchronizing working hours of cross-shore agile teams	Geographical Distribution	yes	Scrum Master	3	[3], [36], [39]
Dealing with geographical distance between agile teams	Geographical Distribution	yes	Initiative-Related	3	[19], [36], [39]
Dealing with lacking team cohesion at different locations	Geographical Distribution	yes	Scrum Master	3	[33], [36], [39]
Building trust of stakeholders in agile practices	Culture & Mindset	no	Scrum Master	3	[3], [41], [52]
Ensuring the reuse of enterprise assets	Enterprise Architecture	yes	Enterprise Architect	3	[28], [47], [53]
Definining clear and visible priorities	Requirements Engineering	no	Product Owner	3	[29], [46], [54]
Establishing automated testing	Quality Assurance	no	Test Team	3	[3], [12], [13]
Creating lightweight documentation	Knowledge Management	no	Development Team	3	[26], [37], [47]

TABLE III
LARGE-SCALE AGILE DEVELOPMENT CHALLENGES (CONTINUED)

Name	Challenge Category	Novelty	Relationship to Stakeholders	Number	Sources
Facilitating standardization across agile teams	Enterprise Architecture	yes	Enterprise Architect	3	[3], [12], [60]
Establishing a culture of continuous improvement	Culture & Mindset	no	Scrum Master, Agile Coach	3	[25], [56], [61]
Applying agile practices for developing or maintaining legacy systems	Software-Architecture	yes	Software Architect	3	[13], [44], [47]
Dealing with unplanned requirements and risks	Project Management	no	Program Manager, Product Owner, Product Manager	3	[31], [33], [37]
Rearranging physical spaces	Tooling	no	Scrum Master	3	[3], [12], [50]
Enforcing customer involvement	Culture & Mindset	no	Product Owner	3	[26], [47], [46]
Dealing with communication gaps with stakeholders	Communication & Coordination	yes	Initiative-Related	3	[26], [49], [56]
Dealing with black and white mindsets	Culture & Mindset	no	Agile Coach	2	[3], [42]
Dealing with closed mindedness	Culture & Mindset	no	Agile Coach	2	[3], [42]
Dealing with higher-level management interferences	Culture & Mindset	no	Scrum Master	2	[28], [58]
Demonstrating the value of architecting	Software-Architecture	yes	Software Architect	2	[45], [58]
Dealing with increased efforts by establishing inter-team communication	Communication & Coordination	yes	Initiative-Related	2	[20], [31]
Dealing with lacking sense of ownership responsibilities for developed services	Culture & Mindset	yes	Initiative-Related	2	[20], [33]
Ensuring that agile teams adhere to architecture-related activities	Enterprise Architecture	yes	Enterprise Architect	2	[28], [53]
Providing agile teams appropriate automation and scalable infrastructure	Tooling	no	Enterprise Architect	2	[17], [33]
Ensuring traceability of tests and requirements	Quality Assurance	no	Test Team	2	[3], [62]
Making a cost and schedule estimation	Project Management	no	Product Owner, Product Manager, Program Manager	2	[26], [34]
Creating a teamwork centric rewarding model	Project Management	no	Initiative-Related	2	[3], [12]
Defining clear roles and responsibilities	Project Management	no	Initiative-Related	2	[12], [46]
Decomposing agile teams in smaller independent teams	Enterprise Architecture	yes	Program Manager, Enterprise Architect	2	[34], [56]
Dealing with loss of management control	Culture & Mindset	no	Initiative-Related	2	[31], [47]
Establishing a common understanding of agile thinking and practices	Methodology	yes	Agile Coach	2	[3], [12]
Creating and estimating user stories	Requirements Engineering	no	Product Owner, Development Team	2	[3], [12]
Dealing with cultural differences between cross-shore agile teams	Geographical Distribution	yes	Scrum Master	2	[19], [49]
Dealing with fixed price contracts in agile software development	Project Management	no	Product Manager, Program Manager	2	[47], [63]
Explaining requirements to stakeholders	Communication & Coordination	no	Development Team	2	[25], [51]
Defining a lightweight formal review process for new technologies	Enterprise Architecture	yes	Enterprise Architect	1	[42]
Dealing with office politics	Culture & Mindset	no	Initiative-Related	1	[42]
Fostering technical excellence	Software-Architecture	yes	Software Architect	1	[58]
Encouraging Development Teams to talk about tasks and impediments	Culture & Mindset	no	Agile Coach, Scrum Master	1	[39]
Writing understandable automated tests	Quality Assurance	no	Test Team	1	[62]
Establishing requirements verification	Requirements Engineering	no	Product Owner	1	[26]
Defining high-level requirements a.k.a. epics	Requirements Engineering	yes	Portfolio Manager, Product Owner	1	[12]
Measuring the success of the large-scale agile development program	Project Management	yes	Product Owner	1	[47]
Considering required competencies when assigning teams to tasks	Project Management	yes	Initiative-Related	1	[31]
Dealing with decreased predictability	Project Management	no	Initiative-Related	1	[47]
Empowering agile teams to make decisions	Culture & Mindset	no	Initiative-Related	1	[29]
Forming and managing autonomous teams	Communication & Coordination	yes	Initiative-Related	1	[12]
Coordinating tests and deployment with external parties	Quality Assurance	no	Test team, Development Team	1	[31]
Establishing a lightweight review process for adopting new technologies	Enterprise Architecture	yes	Enterprise Architect	1	[17]
Building an effective coaching model	Methodology	no	Agile Coach	1	[64]
Synchronizing sprints in the large-scale agile development program	Communication & Coordination	yes	Scrum Master	1	[60]

V. Conclusion and future work

In this study, we presented a structured literature review on recurring challenges of stakeholders and initiatives in large-scale agile development. We analyzed 73 papers, in order to describe reported challenges for large-scale agile development. In total, 79 challenges were identified and grouped into eleven challenge categories. We will extend our preliminary study by collecting data from our large-scale agile development workshops and case studies with industry partners. In parallel, we will perform a structured survey among companies in Germany to demonstrate the applicability of our large-scale agile pattern language, which provides the structure for documenting practice-proven solutions to recurring large-scale agile development challenges. After a huge data collection and evaluating the new pattern language, we will publish the Large-Scale Agile Pattern Catalog containing patterns and challenges.

References

[1] P. Kettunen, "Extending software project agility with new product development enterprise agility," *Software Process: Improvement and Practice*, vol. 12, no. 6, pp. 541–548, 2007.

[2] T. Dingsøyr, S. Nerur, V. Balijepally, and N. B. Moe, "A decade of agile methodologies: Towards explaining agile software development," 2012.

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

[4] M. Alqudah and R. Razali, "A review of scaling agile methods in large software development," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 6, no. 6, pp. 28–35, 2016.

[5] VersionOne, "12th annual state of agile report," VersionOne, Tech. Rep., 2018.

[6] E. Hossain, M. A. Babar, and H.-y. Paik, "Using scrum in global software development: a systematic literature review," in *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*. Ieee, 2009, pp. 175–184.

[7] I. Inayat, S. S. Salim, S. Marczak, M. Daneva, and S. Shamshirband, "A systematic literature review on agile requirements engineering practices and challenges," *Computers in human behavior*, vol. 51, pp. 915–929, 2015.

[8] W. R. Fitriani, P. Rahayu, and D. I. Sensuse, "Challenges in agile software development: A systematic literature review," in *Advanced Computer Science and Information Systems (ICACSIS), 2016 International Conference on*. IEEE, 2016, pp. 155–164.

[9] A. W. Schneider and F. Matthes, "Evolving the eam pattern language," in *Proceedings of the 20th European Conference on Pattern Languages of Programs*. ACM, 2015, p. 45.

[10] J. Vom Brocke, A. Simons, B. Niehaves, K. Riemer, R. Plattfaut, and A. Cleven, "Reconstructing the giant: On the importance of rigour in documenting the literature search process," in *ECIS*, vol. 9, 2009, pp. 2206–2217.

[11] D. S. Cruzes and T. Dyba, "Recommended steps for thematic synthesis in software engineering," in *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on*. IEEE, 2011, pp. 275–284.

[12] M. Paasivaara and C. Lassenius, "Scaling scrum in a large globally distributed organization: A case study," in *Global Software Engineering (ICGSE), 2016 IEEE 11th International Conference on*. IEEE, 2016, pp. 74–83.

[13] U. Viswanath, "Lean transformation: Adapting to the change, factors for success and lessons learnt during the journey: A case study in a multi location software product development team," in *Proceedings of the 9th India Software Engineering Conference*. ACM, 2016, pp. 156–162.

[14] E. Bjarnason, K. Wnuk, and B. Regnell, "A case study on benefits and side-effects of agile practices in large-scale requirements engineering," in *Proceedings of the 1st Workshop on Agile Requirements Engineering*, ser. AREW '11. New York, NY, USA: ACM, 2011, pp. 31–35.

[15] A. Martini, J. Bosch, and M. Chaudron, "Architecture technical debt: Understanding causes and a qualitative model," in *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, Aug 2014, pp. 85–92.

[16] K. Rautiainen, J. von Schantz, and J. Vahaniitty, "Supporting scaling agile with portfolio management: Case paf.com," in *2011 44th Hawaii International Conference on System Sciences*, Jan 2011, pp. 1–10.

[17] R. P. Maranzato, M. Neubert, and P. Herculano, "Moving back to scrum and scaling to scrum of scrums in less than one year," in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. ACM, 2011, pp. 125–130.

[18] T. Dybå and T. Dingsøyr, "Agile project management: From self-managing teams to large-scale development," in *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ser. ICSE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 945–946.

[19] R. Vivian, H. Tarmazdi, K. Falkner, N. Falkner, and C. Szabo, "The development of a dashboard tool for visualising online teamwork discussions," in *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ser. ICSE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 380–388.

[20] E. Moore and J. Spens, "Scaling agile: Finding your agile tribe," in *Agile 2008 Conference*, Aug 2008, pp. 121–124.

[21] N. B. Moe, H. H. Olsson, and T. Dingsøyr, "Trends in large-scale agile development: A summary of the 4th workshop at xp2016," in *Proceedings of the Scientific Workshop Proceedings of XP2016*, ser. XP '16 Workshops. New York, NY, USA: ACM, 2016, pp. 1:1–1:4.

[22] T. Dingsøyr, K. Rolland, N. B. Moe, and E. A. Seim, "Coordination in multi-team programmes: An investigation of the group mode in large-scale agile software development," *Procedia Computer Science*, vol. 121, pp. 123–128, 2017.

[23] K. Crowston, K. Chudoba, M. B. Watson-Manheim, and P. Rahmati, "Inter-team coordination in large-scale agile development: A test of organizational discontinuity theory," in *Proceedings of the Scientific Workshop Proceedings of XP2016*, ser. XP '16 Workshops. New York, NY, USA: ACM, 2016, pp. 2:1–2:5.

[24] S. Bick, A. Scheerer, and K. Spohrer, "Inter-team coordination in large agile software development settings: Five ways of practicing agile at scale," in *Proceedings of the Scientific Workshop Proceedings of XP2016*, ser. XP '16 Workshops. New York, NY, USA: ACM, 2016, pp. 4:1–4:5.

[25] M. Paasivaara, "Adopting safe to scale agile in a globally distributed organization," in *Proceedings of the 12th International Conference on Global Software Engineering*, ser. ICGSE '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 36–40.

[26] K. H. Rolland, "'desperately' seeking research on agile requirements in the context of large-scale agile projects," in *Scientific Workshop Proceedings of the XP2015*, ser. XP '15 workshops. New York, NY, USA: ACM, 2015, pp. 5:1–5:6.

[27] H. Nyrud and V. Stray, "Inter-team coordination mechanisms in large-scale agile," in *Proceedings of the XP2017 Scientific Workshops*, ser. XP '17. New York, NY, USA: ACM, 2017, pp. 16:1–16:6.

[28] A. Martini and J. Bosch, "The danger of architectural technical debt: Contagious debt and vicious circles," in *2015 12th Working IEEE/IFIP Conference on Software Architecture*, May 2015, pp. 1–10.

[29] R. K. Gupta, P. Manikreddy, and K. Arya, "Pragmatic scrum transformation: Challenges, practices & impacts during the journey a case study in a multi-location legacy software product development team," in *Proceedings of the 10th Innovations in Software Engineering Conference*. ACM, 2017, pp. 147–156.

[30] M. A. Babar, "An exploratory study of architectural practices and challenges in using agile software development approaches," in *2009 Joint Working IEEE/IFIP Conference on Software Architecture European Conference on Software Architecture*, Sept 2009, pp. 81–90.

[31] J. E. Hannay and H. C. Benestad, "Perceived productivity threats in large agile development projects," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '10. New York, NY, USA: ACM, 2010, pp. 15:1–15:10.

[32] B. Murphy, C. Bird, T. Zimmermann, L. Williams, N. Nagappan, and A. Begel, "Have agile techniques been the silver bullet for software development at microsoft?" in *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, Oct 2013, pp. 75–84.

[33] M. S. Roopa, C. Sankarasubbiah, and V. S. Mani, "Usable software at the end of each takt: A milestone in the lean transformation of a globally distributed software development team," in *Proceedings of the 12th International Conference on Global Software Engineering*, ser. ICGSE '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 116–120.

[34] M. Kircher and P. Hofman, "Combining systematic reuse with agile development: Experience report," in *Proceedings of the 16th International Software Product Line Conference - Volume 1*, ser. SPLC '12. New York, NY, USA: ACM, 2012, pp. 215–219.

[35] D. Rosenberg, B. Boehm, B. Wang, and K. Qi, "Rapid, evolutionary, reliable, scalable system and software development: The resilient agile process," in *Proceedings of the 2017 International Conference on Software and System Process*, ser. ICSSP 2017. New York, NY, USA: ACM, 2017, pp. 60–69.

[36] R. Sindhgatta, B. Sengupta, and S. Datta, "Coping with distance: An empirical study of communication on the jazz platform," in *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, ser. OOPSLA '11. New York, NY, USA: ACM, 2011, pp. 155–162.

[37] M. Budwig, S. Jeong, and K. Kelkar, "When user experience met agile: A case study," in *CHI '09 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '09. New York, NY, USA: ACM, 2009, pp. 3075–3084.

[38] F. O. Bjørnson and K. Vestues, "Knowledge sharing and process improvement in large-scale agile development," in *Proceedings of the Scientific Workshop Proceedings of XP2016*, ser. XP '16 Workshops. New York, NY, USA: ACM, 2016, pp. 7:1–7:5.

[39] M. Paasivaara, S. Durasiewicz, and C. Lassenius, "Distributed agile development: Using scrum in a large project," in *2008 IEEE International Conference on Global Software Engineering*, Aug 2008, pp. 87–95.

[40] K. H. Rolland, "Scaling across knowledge boundaries: A case study of a large-scale agile software development project," in *Proceedings of the Scientific Workshop Proceedings of XP2016*, ser. XP '16 Workshops. New York, NY, USA: ACM, 2016, pp. 5:1–5:5.

[41] I. Therrien and E. LeBel, "From anarchy to sustainable development: Scrum in less than ideal conditions," in *2009 Agile Conference*, Aug 2009, pp. 289–294.

[42] A. Mahanti, "Challenges in enterprise adoption of agile methods," 2006.

[43] M. Paasivaara, C. Lassenius, and V. T. Heikkilä, "Inter-team coordination in large-scale globally distributed scrum: Do scrum-of-scrums really work?" in *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*. ACM, 2012, pp. 235–238.

[44] B. Boehm and R. Turner, "Management challenges to implementing agile processes in traditional development organizations," *IEEE Software*, vol. 22, no. 5, pp. 30–39, Sept 2005.

[45] P. Abrahamsson, M. A. Babar, and P. Kruchten, "Agility and architecture: Can they coexist?" *IEEE Software*, vol. 27, no. 2, pp. 16–22, March 2010.

[46] H. Ayed, B. Vanderose, and N. Habra, "Supported approach for agile methods adaptation: An adoption study," in *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, ser. RCoSE 2014. New York, NY, USA: ACM, 2014, pp. 36–41.

[47] P. Rodríguez, J. Markkula, M. Oivo, and K. Turula, "Survey on agile and lean usage in finnish software industry," in *Empirical Software Engineering and Measurement (ESEM), 2012 ACM-IEEE International Symposium on*. IEEE, 2012, pp. 139–148.

[48] J. Pries-Heje and M. M. Krohn, "The safe way to the agile organization," in *Proceedings of the XP2017 Scientific Workshops*, ser. XP '17. New York, NY, USA: ACM, 2017, pp. 18:1–18:3.

[49] P. Lous, M. Kuhrmann, and P. Tell, "Is scrum fit for global software engineering?" in *Proceedings of the 12th International Conference on Global Software Engineering*, ser. ICGSE '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 1–10.

[50] M. Hallikainen, "Experiences on agile seating, facilities and solutions: Multisite environment," in *2011 IEEE Sixth International Conference on Global Software Engineering*, Aug 2011, pp. 119–123.

[51] O. Ktata and G. Lévesque, "Agile development: Issues and avenues requiring a substantial enhancement of the business perspective in large projects," in *proceedings of the 2nd Canadian conference on computer science and software engineering*. ACM, 2009, pp. 59–66.

[52] D. Wilby, "Roadmap transformation: From obstacle to catalyst," in *2009 Agile Conference*, Aug 2009, pp. 229–234.

[53] M. Rizwan and J. Qureshi, "Agile software development methodology for medium and large projects," *IET Software*, vol. 6, no. 4, pp. 358–363, August 2012.

[54] D. Talby and Y. Dubinsky, "Governance of an agile software project," in *Proceedings of the 2009 ICSE Workshop on Software Development Governance*, ser. SDG '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 40–45.

[55] R. Vallon, C. Dräger, A. Zapletal, and T. Grechenig, "Adapting to changes in a project's dna: A descriptive case study on the effects of transforming agile single-site to distributed software development," in *2014 Agile Conference*, July 2014, pp. 52–60.

[56] B. Sheth, "Scrum 911! using scrum to overhaul a support organization," in *2009 Agile Conference*, Aug 2009, pp. 74–78.

[57] M. Laanti, "Implementing program model with agile principles in a large software development organization," in *2008 32nd Annual IEEE International Computer Software and Applications Conference*, July 2008, pp. 1383–1391.

[58] S. Angelov, M. Meesters, and M. Galster, "Architects in scrum: What challenges do they face?" 11 2016, pp. 229–237.

[59] D. Broschinsky and L. Baker, "Using persona with xp at landesk software, an avocent company," in *Agile 2008 Conference*, Aug 2008, pp. 543–548.

[60] V. Sithole and F. Solms, "Synchronized agile," in *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists*, ser. SAICSIT '16. New York, NY, USA: ACM, 2016, pp. 39:1–39:9.

[61] P. Rodríguez, K. Mikkonen, P. Kuvaja, M. Oivo, and J. Garbajosa, "Building lean thinking in a telecom software development organization: Strengths and challenges," in *Proceedings of the 2013 International Conference on Software and System Process*, ser. ICSSP 2013. New York, NY, USA: ACM, 2013, pp. 98–107.

[62] T. M. King, G. Nunez, D. Santiago, A. Cando, and C. Mack, "Legend: An agile dsl tot booktitle = Proceedings of the 2014 International Symposium on Software Testing and Analysis, series = ISSTA 2014, year = 2014, isbn = 978-1-4503-2645-2, location = San Jose, CA, USA, pages = 409–412, numpages = 4, acmid = 2628048, publisher = ACM, address = New York, NY, USA, keywords = Agile Development, Behavior-Driven Development, Domain-Specific Languages, Software Testing, Test Automation,."

[63] P. Mohagheghi and M. Jørgensen, "What contributes to the success of it projects? success factors, challenges and lessons learned from an empirical study of software projects in the norwegian public sector," in *Software Engineering Companion (ICSE-C), 2017 IEEE/ACM 39th International Conference on*. IEEE, 2017, pp. 371–373.

[64] S. Hanly, L. Wai, L. Meadows, and R. Leaton, "Agile coaching in british telecom: making strawberry jam," in *AGILE 2006 (AGILE'06)*, July 2006, pp. 9 pp.–202.

[65] "The scrum guide," http://www.scrumguides.org/scrum-guide.html, accessed: 2017-04-12.

Documenting Recurring Concerns and Patterns in Large-Scale Agile Development

Ömer Uludağ
Technische Universität München
Garching bei München, Germany
oemer.uludag@tum.de

Nina-Mareike Harders
Technische Universität München
Garching bei München, Germany
nina.harders@tum.de

Florian Matthes
Technische Universität München
Garching bei München, Germany
matthes@tum.de

Abstract

The introduction of agile methods at scale entails unique concerns such as inter-team coordination, dependencies to other organizational units, or distribution of work without a defined architecture. Compared to the rich body of agile software development literature describing typical challenges and best practices, recurring concerns and patterns in large-scale agile development are not yet documented extensively. We aim to fill this gap by presenting a pattern language for large-scale agile software development as part of our larger research initiative in close collaboration with 10 companies. The structure and practical relevance of the proposed language were evaluated by 14 interviews. In this paper, we showcase our pattern language by presenting four patterns.

CCS Concepts

• **Software and its engineering → Agile software development**.

Keywords

concerns, large-scale agile development, patterns

ACM Reference Format:
Ömer Uludağ, Nina-Mareike Harders, and Florian Matthes. 2019. Documenting Recurring Concerns and Patterns in Large-Scale Agile Development. In *Proceedings of ACM Conference (EuroPLoP'19).* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3361149.3361176

1 Introduction

Over the past two decades, software development has experienced substantial growth in the use of agile methods [Maiden and Jones 2010]. Unlike traditional methods that focus on upfront plans and documentation, agile methods such as Extreme Programming and Scrum strongly encourage team collaboration, change tolerance, evolutionary delivery, and active customer involvement [Dingsøyr and Moe 2014; Kettunen 2007]. The fundamental assumptions are that small, self-organizing teams develop adaptive software using the principles of continuous design improvement and testing based on rapid iterations and frequent feedback loops [Nerur et al. 2005]. Hitherto, agile methods were mostly applied within the so-called "agile sweet spot": small, co-located teams of less than 15 people doing greenfield development for non-safety-critical systems in a volatile environment [Kruchten 2013; Nord et al. 2014]. Their proven and potential benefits made them also attractive for projects outside of the sweet spot [Dikert et al. 2016]. Thus, there is an industry trend towards introducing agile methods at scale [Dikert et al. 2016; VersionOne 2018]. The term 'large-scale agile development' is used to describe multi-team development efforts that make use of agile principles involving a high number of actors and interfaces with existing systems [Dingsøyr and Moe 2014; Rolland et al. 2016]. However, the adoption of agile methods at larger scale entails organizations with unprecedented challenges such as general resistance to change, coordination challenges in multi-team environments, and dependencies to other existing environments [Dikert et al. 2016; Uludağ et al. 2018]. Compared to the rich body of agile software development literature describing typical challenges (cf. [Hossain et al. 2009; Inayat et al. 2015]) and best practices (cf. [Beedle et al. 2010, 1999; Coplien and Harrison 2004; ScrumPLoP 2019]), the documentation of concerns and patterns in large-scale agile development is still scarce. Our study is inspired by the pattern-based approach to Enterprise Architecture Management (EAM) [Ernst 2010; Schneider and Matthes 2015] and aims to fill this gap by providing best practices for recurring concerns of stakeholders in large-scale endeavors. As a starting point, we introduce the concept of large-scale agile development patterns and present four patterns that exemplarily demonstrate the proposed language.

The remainder of this paper is structured as follows. In Section 2, we present the research approach that follows the pattern-based design research (PDR) method. In Section 3, we provide an overview of related works in the field of large-scale agile development and describe related pattern languages. We elaborate the proposed large-scale agile development pattern language in Section 4. In Section 5, we present four patterns. Section 6 shows corresponding evaluation results. In Section 7, we conclude our paper with a summary of our results and remarks on future research.

2 Research Approach

Our research initiative aims to document best practices that address concerns in large-scale agile development. To balance the rigor and relevance of the research, we followed the pattern-based design research (PDR) method as recommended by [Buckl et al. 2013]. The PDR method enables researchers to theorize and learn from the intervention at the industry partners while performing rigorous and relevant design science research. It builds on established concepts such as patterns and design theories. As depicted in Fig. 1, the PDR method consists of four phases: *observe & conceptualize*, *pattern-based theory building & nexus instantiation*, *solution design & application*, and *evaluation & learning*. Within the *observe & conceptualize* phase, good practices for recurring concerns are

EuroPLoP '19, July 3–7, 2019, Irsee, Germany

ACM ISBN 978-1-4503-6206-1/19/07…$15.00
https://doi.org/10.1145/3361149.3361176

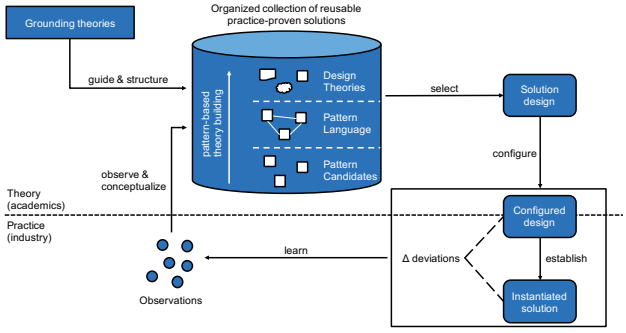


Figure 1: Pattern-based design research [Buckl et al. 2013]

observed and documented following a typical pattern structure (see Section 4). These pattern candidates are then conceptualized by using grounding theories and evolve into actual patterns by fulfilling the *rule of three*[1] [Coplien 1996], which are then integrated into the large-scale agile development pattern language. Design theories can be developed by documenting appropriate context and problem descriptions. Pattern candidates, patterns, the pattern language, and design theories together form an organized collection of reusable, proven solutions. Within the *solution design & application* phase, typical stakeholders in large-scale agile development use this knowledge base and select patterns based on their individual concerns. The selected pattern must be configured and adapted to the terminology of the company. After its configuration, the pattern can be established within the case company. During the *evaluation & learning* phase, deviations between the actual and original pattern configuration are detected and documented, which can be used to identify new best practices.

3 Related Work and Pattern Languages

According to Version One's 12th survey on the state of agile, companies are increasingly applying agile methods to large-scale projects [VersionOne 2018]. 52% of all respondents worked in organizations where more than half of the development teams worked with agile methods [VersionOne 2018]. Despite the relevance of this topic for practitioners, sound academic research is lacking, especially regarding to challenges and success factors [Dikert et al. 2016]. Some researchers recognized this gap and started to publish scientific papers, which are described below.

[Dikert et al. 2016] made a first attempt by conducting a systematic literature review of industrial large-scale agile transformations. They presented qualitative findings describing 35 reported challenges and 29 success factors from 42 different organizations. Challenge categories that received most attention were *agile difficult to implement*, *integrating non-development functions*, *change resistance*, and *requirements engineering challenges*. The most salient success factors were *management support*, *choosing and customizing the agile model*, *training and coaching*, and *mindset and alignment*. By means of a literature review, [Kalenda et al. 2018] identified practices, challenges, and success factors of large companies adopting agile methods. These findings were then compared and used to study a software company that was in the process of scaling agile methods. They identified four challenges, namely *resistance to change*, *quality assurance issues*, *integrating with non-agile parts of the organization*, and *too fast roll-out*. In addition, they determined four success factors: *unification of views and values*, *executive sponsorship and management support*, *company culture*, and *prior agile and lean experience*. In a previous study, we identified typical concerns of stakeholders and initiatives in large-scale agile development based on a structured literature review [Uludağ et al. 2018]. Based on the analysis of 73 papers, we identified 14 typical stakeholders in large-scale agile development, e.g., *development team*, *scrum master*, and *software architect*. In a subsequent step, we revealed typical concerns of respective stakeholders. In total, 79 challenges were identified and grouped into eleven challenge categories, which include *culture and mindset*, *enterprise architecture*, and *geographical distribution*, to name a few [Uludağ et al. 2018]. Our previous work constitutes the foundation of this paper since it also identified pattern candidates of some of our pattern language elements: *stakeholders*, *challenges*, and candidates for *methodology patterns*, *architecture principles*, *viewpoint patterns*, and *anti-patterns* [Uludağ et al. 2018]. Speaking of which, [Meszaros and Doble 1997] recommend reading other related pattern languages while writing patterns. By doing that, we identified some related pattern languages (see Table 1). Based on the comparison of related pattern languages, we identified the following shortcomings, which we aim to address with our proposed large-scale agile development pattern language:

- Only 10 out of 507 identified "potentially relevant" patterns focus on large-scale agile development.
- Our evaluation results in Section 6 show that practitioners ask for pattern languages that categorize patterns in the way they are executed. However, the analyzed pattern languages do not necessarily meet these expectations.
- The results of our evaluation in Section 6 show that a pattern language should include related stakeholders who apply patterns to address their concerns. Nevertheless, the concept of stakeholders is yet neglected, making the pattern language less practical for practitioners as they try to find relevant patterns for their specific roles as quickly and easily as possible.

4 Large-Scale Agile Development Pattern Overview

The application of agile methods on a large scale brings along unique challenges and difficulties [Boehm and Turner 2005; Dikert et al. 2016], e.g., increased number of stakeholders and, coordination complexity, and difficult architectural integration [Badampudi et al. 2013; Paasivaara and Lassenius 2014]. Tackling these is the key to reap the full benefits of agility in large-scale settings [Kettunen and Laanti 2008]. There are several scaling agile frameworks that pledge to resolve the aforementioned issues but are still in a nascent state [Alqudah and Razali 2016; Dingsøyr et al. 2019]. But even valuable research studies that provide explanations of how to address the challenges of large-scale agile development remain

[1] The *rule of three* states that a documented pattern must refer to at least three known uses in practice to ensure the re-usability of the provided solution.

Table 1: Overview of Related Pattern Languages

Source	Scope & goal	Focus on agile development	Number of patterns	Pattern categories	Pattern examples
[Coplien 1995]	Collection of patterns for shaping a new organization and its development processes	Partially	42	(1) Process patterns; (2) Organizational patterns	- Code Ownership - Gatekeeper - Fire Walls
[Harrison 1996]	Collection of patterns for creating effective software development teams	No	4	–	- Unity of Purpose - Diversity of Membership - Lock 'Em Up Together
[Beedle et al. 1999]	Collection of Scrum patterns	Yes	3	–	- Sprint - Backlog - Scrum Meetings
[Taylor 2000]	Collection of patterns for creating product software development environments	No	9	(1) Establishing a Production Potential; (2) Maintaining a Production Potential; (3) Preserving a Production Potential	- Deliverables to Go - Pulse - Bootstrapping
[Coplien and Harrison 2004]	Collection of organizational patterns that are combined into a collection of four pattern languages	Yes	94	(1) Project Management; (2) Piecemeal Growth; (3) Organizational Style; (4) People and Code	- Skill Mix - Demo Prep - Few Roles
[Elssamadisy 2008]	Collection of patterns for successfully adopting agile practices	Yes	38	(1) Feedback Practices; (2) Technical Practices; (3) Supporting Practices; (4) The Clusters	- Refactoring - Continuous Integration - Simple Design
[Beedle et al. 2010]	Collection of the most essential best practices of Scrum	Yes	11	–	- Daily Scrum - Sprint Backlog - Sprint Review
[Välimäki 2011]	Enhancing performance of project management work through improved global software project management practice	Partially	18	(1) Directing a Project; (2) Starting up a Project; (3) Initiating a Project; (4) Controlling a Stage; (5) Managing Stage Boundaries; (6) Closing a Project; (7) Managing Product Delivery; (8) Planning	- Collocated Kick-Off - Choose Roles in Sites - Iteration Planning
[Mitchell 2016]	Collection of patterns to address agile transformation problems	Yes	54	(1) Patterns of Method; (2) Patterns of Responsibility; (3) Patterns of Representation; (4) Anti-Patterns	- Limited WIP - Kanban Sandwich - Controlled Failure
[ScrumPLoP 2019]	Body of pattern literature around agile and Scrum communities	Yes	234 (10)	(1) Value Stream; (2) Team; (3) Sprint; (4) Process Improvement; (5) Product Organization; (6) Distributed Scrum; (7) Scaling Scrum; (8) Scrum Core; (9) Misc	- Scrum Master - Scrum of Scrums - Portfolio Standup

scarce [Bick et al. 2018]. Thus, following the idea of [Alexander 1977], the identification of recurring concerns and documentation of best practices in this context seems to be useful. Subsequently, we will introduce the structure of our pattern language (see Fig. 2). The pattern language distinguishes between three different types of patterns:

- **Coordination Patterns (C-Patterns)** define coordination mechanisms to address recurring coordination concerns, i.e., managing dependencies between activities, tasks or resources.
- **Methodology Patterns (M-Patterns)** define concrete steps to be taken to address given concerns.
- **Viewpoint Patterns (V-Patterns)** define proven ways to visualize information in form of documents, boards, metrics, models, and reports in order to address recurring concerns.

In addition, the pattern language includes four additional concepts:

- **Stakeholders** (in our context) are all persons who are actively involved in, have an interest in or are in some way affected by large-scale agile development [Uludağ et al. 2018].
- **Concerns** can manifest themselves in many forms, e.g., goals, responsibilities or risks [42010:2011(E) 2011].
- **Principles** are enduring and general guidelines that address given concerns by providing a common direction for action.
- **Anti-Patterns (A-Patterns)** describe typical mistakes and present revised solutions, which help pattern users to prevent these pitfalls.

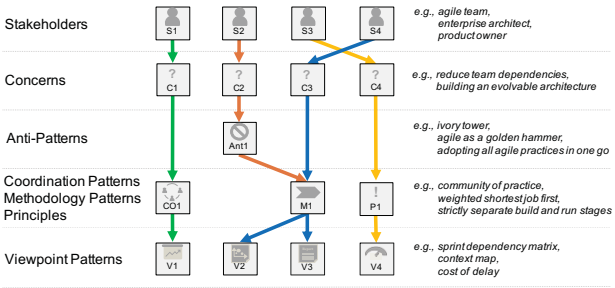


Figure 2: Conceptual overview of the proposed pattern language

Fig. 3 depicts the current version of our large-scale agile development pattern language, which can also be found on our prototypical web application[2]. The four highlighted nodes in Fig. 3 represent the four patterns that will be presented in Section 5. A detailed listing of the large-scale agile development patterns and concepts can be

[2]https://scaling-agile-hub.sebis.in.tum.de/#/patterns

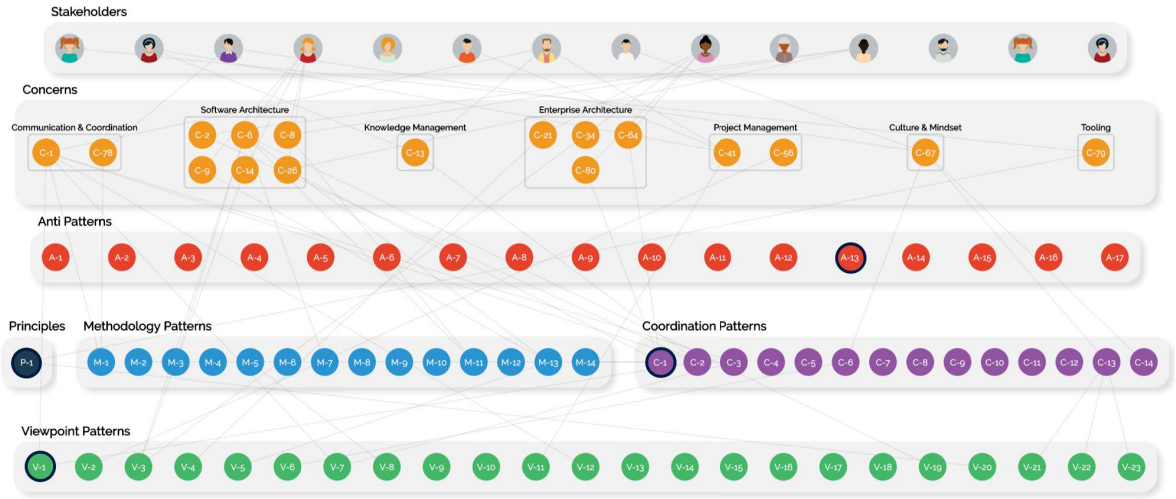


Figure 3: Current version of the large-scale agile development pattern language *

found in Appendix B.

Popular pattern forms include, among others, the Alexandrian Form, Gang of Four Form, Coplien Form, and Fowler Form [Ernst 2010; Fowler 2006]. All have specific benefits and limitations depending on the context [Ernst 2010]. Since there is no ideal pattern form, the author must consider his / her experience, the intention, and target audience when selecting either an existing form or creating a new one [Buschmann et al. 2007b; Ernst 2010]. According to [Fowler 2006], this choice is a personal decision and should also consider one's writing style and the ideas to be conveyed. Large-scale agile development patterns follow a template similar to [Buschmann et al. 1996; Ernst 2010]. Fig. 4 depicts the meta-model and the key

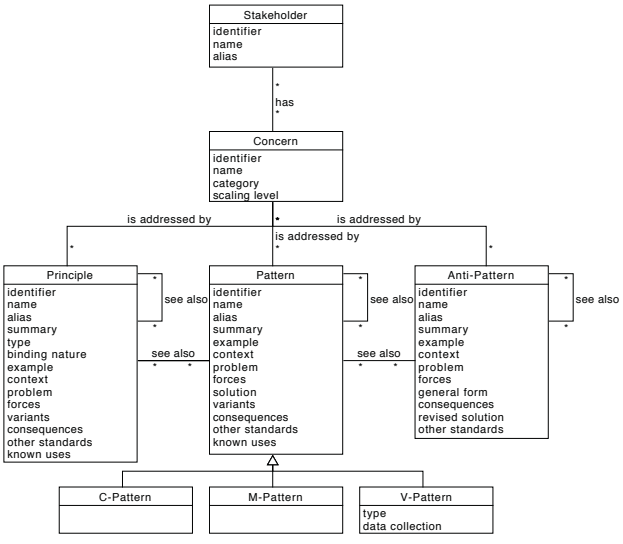


Figure 4: Meta-model of the proposed pattern language

elements used to document the concepts and patterns of the pattern language. All elements have an *identifier* and *name* which simplify referencing. A stakeholder has an additional section called *alias* that contains a list of synonyms and related role names. A concern has two additional sections called *category* and *scaling level* which denote the category and at which organizational level a concern occurs. Besides identifiers and names, principles, patterns, and anti-patterns consist of eight common sections: the *problem* and *context* sections describe problems and situations to or in which they apply. The *forces* section describes why the problem is difficult to solve. *summary* shortly recapitulates the solution. The *consequences* section contains associated benefits and liabilities, while the optional *other standards* and *see also* sections provide references to other solutions and frameworks. The *alias* section provides a list of synonyms. The *example* section illustrates the problem to be addressed. Principles and patterns consist of *variants* and *known uses* sections showing variants and alternatives as well as proven applications in practice. The *type* and *binding nature* sections are unique to principles and indicate their topic and whether they are recommended or mandatory. The *solution* section explains the recommended solution for a pattern. Specific to anti-patterns, the *general form* and *revised solution* sections include the recurring, not working solution and a revised solution presented. V-Patterns have *type* and *data collection* sections which show the visualization concept and collection processes required for their creation. Similar to [Buschmann et al. 2007a], we label our patterns with the star notation to denote our level of confidence in the pattern's maturity. Two stars mean that the pattern effectively addresses a genuine problem in its current form. One star denotes that the pattern addresses a real problem but needs to mature. No stars indicate that the pattern is a useful solution to an observed problem but requires significant revision.

We will showcase in the following four patterns in order to highlight the differences between the presented pattern types and concepts, namely Strictly Separate Build and Run Stages (representing Principles), Community of Practice (showing C-Patterns), Iteration Dependency Matrix (demonstrating (V-Patterns), and Don't Use Agile as a Golden Hammer (illustrating A-Patterns).

5 Exemplifying the Large-Scale Agile Development Pattern Language

5.1 Principle: Strictly Separate Build and Run Stages (P-1) *

Principle Overview	
Alias	Build and Run
Summary	Strictly Separate Build and Run Stages ensures that an application's deployment phases are clearly separated.
Type	Software Architecture
Binding Nature	Recommended

5.1.1 Example

For one and a half years, four agile teams of RetailCo have been developing a cloud-based e-commerce platform. However, over the last few iterations, the operability and stability of the e-commerce platform of RetailCo have deteriorated dramatically. The agile teams of RetailCo have difficulties in delivering new versions of the platform in time. The platform also shows long downtimes due to massive traffic at bigger shopping-events such as Black Friday. Moreover, the product owner of the e-commerce platform receives customer complaints due to several bugs in the purchasing process.

5.1.2 Context

The release of stable and reliable cloud-native platforms in large-scale agile development is difficult because multiple agile teams work in parallel on the same software in complex setups aiming to release it as quickly and frequently as possible.

5.1.3 Problem

The following concern is addressed by Strictly Separate Build and Run Stages:

- *How can a cloud-native application be developed in a stable and timely manner?*

5.1.4 Forces

The following forces influence Strictly Separate Build and Run Stages:

- The code of a cloud-native application is changed at run-time, thus, is not reproducible.
- The build, run, and release phases of the deployment process of a cloud-native application are not self-contained, so the entire deployment workflow must be triggered.
- Deviations between the codes in the execution and staging environment are not traceable.

5.1.5 Variants

A possible variant for Strictly Separate Build and Run Stages would be to add a design step before the build stage. Within the design step, a high-level design of the upcoming small feature is created with every iteration. This can help to understand the dependencies of the application such as existing libraries the application is going to use.

5.1.6 Consequences

The following benefits of Strictly Separate Build and Run Stages are known:

- Traceability and reproducibility of releases
- Rollback to previous releases
- Faster releases of codebases to production
- Building a stable and reliable application
- No code is deployed without testing

The following liabilities of Strictly Separate Build and Run Stages are known:

- High degree of automation
- Deployment process is complicated

5.1.7 See Also

In order to measure the level of adherence of agile teams with Strictly Separate Build and Run Stages, the following V-Patterns should be considered:

- Deployment Time
- Deployment Frequency
- Change Frequency
- Mean Time to Change

5.1.8 Other Standards

Strictly Separate Build and Run Stages is also suggested by the Twelve-Factor App [Wiggins 2017].
Strictly Separate Build and Run Stages is extended by the Beyond the Twelve-Factor App [Hoffman 2016].

5.1.9 Known Uses

The following uses of Strictly Separate Build and Run Stages are known:

- GlobalInsureCo
- CarCo
- ITCo
- RetailCo
- PublicInsureCo

5.2 C-Pattern: Community of Practice (C-1) *

C-Pattern Overview	
Alias	Community, Guild
Summary	A Community of Practice are groups of people who share a concern, a set of problems, or a passion about a topic, and who deepen their knowledge and expertise in this area by interacting on an ongoing basis [Wenger et al. 2002].

5.2.1 Example

A vehicle dynamics development department of CarCo aims to transform its current traditional matrix organization to an agile organization by launching a large-scale endeavor with seven agile teams and more than 100 involved stakeholders. During this transformation process, CarCo has difficulties in aligning the agile teams working in the same department as they have no regular meetings on discussing common topics. Furthermore, the software architects of the large-scale agile endeavor recognize that the agile teams use some tools that are incompatible with each other making the integration of their sub products nearly impossible.

5.2.2 Context

Traditional agile approaches such as Scrum do not offer support large-scale cross-team coordination. Thus, establishing efficient coordination and knowledge sharing mechanisms between agile teams as well as between the experts in the teams might be difficult without having suitable knowledge sharing forums.

5.2.3 Problem

The following concern is addressed by Community of Practice:

- *How to create a platform for active knowledge sharing and discussion?*

5.2.4 Forces

The following forces influence Community of Practice:

- Facilitating shared context and knowledge across the organization is difficult
- Internal silos create gaps in knowledge and communication between agile teams

5.2.5 Solution

A Community of Practice meet regularly for knowledge sharing about a specific domain [Wenger et al. 2002]. The focus is to talk about practices that are applied and not to discuss theories. The participants of a Community of Practice are typically not from the same team, but from many different teams all across the organization. In the best case, many different practices can be presented and discussed, leading to a wide knowledge base. The participation in a Community of Practice is usually voluntary. In contrast to the M-Pattern Empowered Community of Practice, a traditional Community of Practice is not able to make binding decisions for the organization.

5.2.6 Variants

A Community of Practice can be set up for a variety of domains. A Community of Practice have been identified in the following domains: Architecture, Testing, Interfaces, Deployments, Leadership, and Infrastructure. In addition, a Community of Practice can also have some decision-making power for different topics, which is described in the M-Pattern Empowered Community of Practice.

5.2.7 Consequences

The following benefits of Community of Practice are known:

- Encouraging knowledge sharing for diverse topics
- Breaking up silos
- Enabling a culture of continuous improvement

The following liabilities of Community of Practice are known:

- Requiring an active involvement of participants
- Topics in the agenda could be too diverse and broad
- Providing right incentives to the participants is challenging

5.2.8 See Also

Community of Practice may be utilized in combination with the following M-Patterns:

- Consensus-Based Decision Making
- Empowered Community of Practice

5.2.9 *Other Standards*

Community of Practice is also recommended and practiced by the following scaling agile frameworks:

- Disciplined Agile Delivery [Ambler and Lines 2012]
- Large-Scale Scrum [Larman and Vodde 2016]
- Scaled Agile Framework [Scaled Agile 2019b]
- Spotify Model [Kniberg and Ivarsson 2012]

In addition, the Community of Practice is also described by [Coplien and Harrison 2004] as Community of Trust.

5.2.10 *Known Uses*

The following uses of Community of Practice are known:

- GlobalInsureCo
- CarCo
- ITCo
- RetailCo
- PublicInsureCo

5.3 V-Pattern: Iteration Dependency Matrix (V-1) *

V-Pattern Overview	
Alias	Sprint Dependency Matrix, Program Board
Summary	Iteration Dependency Matrix visualizes dependencies among teams for the upcoming iterations.
Type	Board

5.3.1 *Example*

The program manager and solution architect of RetailCo's e-commerce platform of RetailCo want to coordinate four agile teams for the five upcoming iterations. Thereby, they need a clear idea of which features or enablers will be done by which agile team in which iteration. In addition, they want to identify cross-team dependencies which might impact their delivery.

5.3.2 *Context*

In a Common Planning, cross-team dependencies between agile teams has to be detected and managed.

5.3.3 *Problem*

The following concerns are addressed by Iteration Dependency Matrix:

- *How to visualize dependencies between agile teams?*
- *How to coordinate multiple agile teams that work on the same product?*
- *How to consider integration issues and dependencies with other subsystems and teams?*

5.3.4 *Forces*

The following forces influence Iteration Dependency Matrix:

- Agile teams that work in the same large-scale agile development program have to be coordinated, i.e., delivering on the same cadence, timing the release of different features, and managing dependencies.
- Some dependencies between agile teams are not immediately visible.

5.3.5 *Solution*

Iteration Dependency Matrix visualizes dependencies between agile teams working on the same product for future iterations. The exemplary visualization in Fig. 5 shows team names as vertical headings, while iteration names are shown as horizontal headings. The blue rectangles represent features, while enablers are depicted as yellow rectangles. Important program milestones or events are depicted as orange rectangles. Each enabler or feature belongs to one team and one iteration. A feature may depend on other enablers. These dependencies are indicated by red strings.

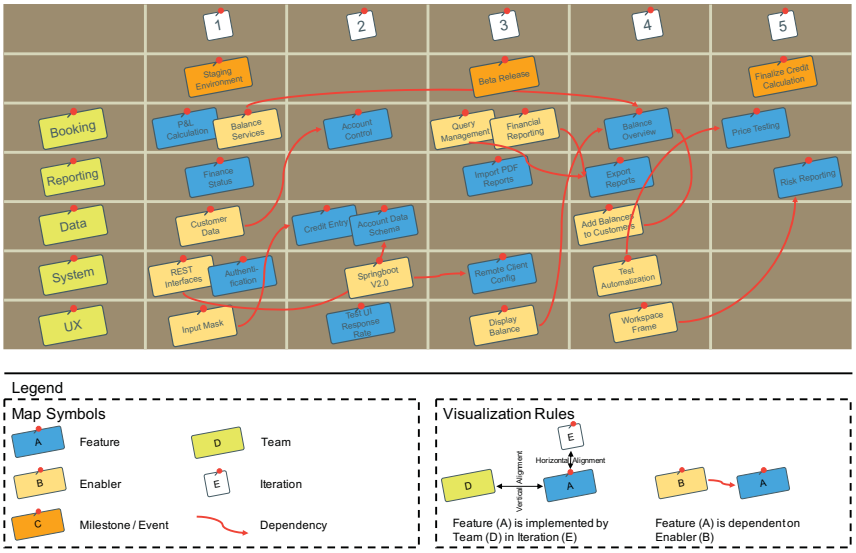


Figure 5: Exemplary view for Iteration Dependency Matrix

The underlying information model of Iteration Dependency Matrix is shown in Fig. 6.

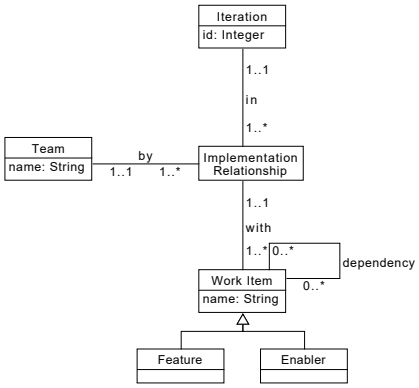


Figure 6: Underlying information model of Iteration Dependency Matrix

5.3.6 Variants

Additional variants exist for Iteration Dependency Matrix. First, enablers may be omitted if the organization does not make a differentiation between enablers and features, i.e., for budgetary reasons (see Fig. 7(b)). However, this variant is not advised as the importance of architectural improvements might be neglected. In addition, the Iteration Dependency Matrix could be simplified by omitting milestones or events that would happen in the upcoming iterations (see Fig. 7(b)). The Iteration Dependency Matrix can also be visualized digitally by the use of software development team collaboration tools. This variant has the advantage that useful information is stored in a digital format and that it can also be used in a distributed Common Planning.

5.3.7 Consequences

The following benefits of Iteration Dependency Matrix are known:

- Visualizing and tracking inter-team dependencies
- Revealing unplanned risks and dependencies
- Providing a visual overview of work to be done
- Optimizing development flow

The following liabilities of Iteration Dependency Matrix are known:

- High manual effort during its creation
- After the Common Planning, it might be abandoned
- Less effective when agile teams are remote

5.3.8 Data Collection

- *Frequency:* Features, enablers, and goals are usually written throughout the development process.
- *Responsible person:* Responsible persons are e.g., the business analyst, product manager, product owner, program manager or solution architect.
- *Data source:* Software development team collaboration tools.
- *Classes to be documented:* Team, enabler, feature, and iteration.

5.3.9 See Also

Iteration Dependency Matrix is used by the following C-Pattern:

- Common Planning

5.3.10 Other Standards

Iteration Dependency Matrix is also known as the Program Board in the Scaled Agile Framework [Scaled Agile 2019a].

5.3.11 Known Uses

The following uses of Iteration Dependency Matrix are known:

- CarCo
- RailCo
- BankingITCo
- RetailCo

- IndustryCo

(a) BankingITCo (b) RetailCo

Figure 7: Two observed Iteration Dependency Matrices

5.4 A-Pattern: Don't Use Agile as a Golden Hammer (A-13) *

A-Pattern Overview	
Alias	Law of the Instrument, Law of the Hammer, Maslow's Hammer, One Size Fits All
Summary	Don't Use Agile as a Golden Hammer shows why it is not advisable to use agile methods in order to solve many kinds of problems.

5.4.1 Example

Success stories of the autonomous driving department with the application of agile methods reached the IT department of LuxCarsCo. The IT management decided to transform all current IT projects to agile.

5.4.2 Context

Agile methods have become very popular in the last few years. Deciding when agile methods are appropriate for a particular project is a difficult task as different aspects have to be considered, e.g., project size, project objectives and requirements, project team, technological realization, and so on.

5.4.3 Problem

The following concerns are addressed by Don't Use Agile as a Golden Hammer:

- *How to choose the correct software development approach?*
- *How to decide whether agile methods should be used for a given project?*

5.4.4 Forces

The following forces occur in the context of Don't Use as a Golden Hammer:

- Several successes tempt the organization to solely use agile methods
- Large investment has been made in agile training
- Lack of motivation to explore alternative methods

5.4.5 General Form

A software development team has gained a lot of attention within the organization due to its success by using agile methods. As a result, the organization believes that every new product should be developed by the use of agile methods. In some cases, the use of agile methods will not solve the problem.

5.4.6 Variants

A possible variant for Don't Use Agile as a Golden Hammer is that not only the management of an organization forces the use of agile methods for software projects, but also existing software development teams obsessively operate use of agile methods.

5.4.7 Consequences

The following benefits of Don't Use Agile as a Golden Hammer are known:

- Management-Buy-In

The following liabilities of Don't Use Agile as a Golden Hammer are known:

- Failed projects

5.4.8 Revised Solution

The most important aspect of the revised solution is: Try to avoid Don't Use Agile as a Golden Hammer. This can be done by using evidence why the use of agile methods might not be appropriate, i.e., in simple projects in which the technological realization and requirements are known (see Fig. 8). Don't Use Agile as a Golden Hammercan also be avoided when a conscious effort towards the exploration of alternative software development methods is made. Use Community of Practice to facilitate the exchange of ideas and experiences and to understand their rationale for applying agile methods.

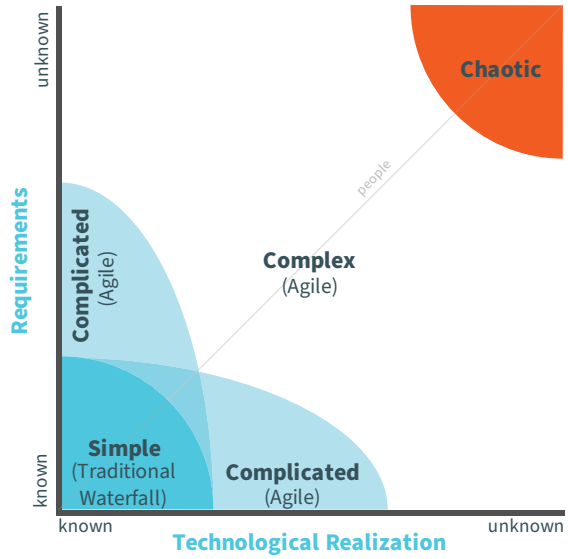


Figure 8: Stacey Matrix [Stacey 1996] adapted to software development

6 Evaluation

We interviewed 14 large-scale agile development experts from 10 organizations to assess the structure and practical relevance of the proposed pattern language. The companies are mainly headquartered in Germany, predominantly active in the IT, insurance, and retail sector, and employ around 2,000 to 200,000 people. Most of them started with large-scale agile development a few years ago. We prepared a questionnaire (see Appendix A) containing scale response (on a five-point Likert scale) and open-ended questions to collect expert feedback on our proposed pattern language. Table 2 lists the roles and professional experiences of all interviewed experts with large-scale agile development. During the interviews,

Table 2: Interview partners for evaluating the proposed pattern language

No	Alias	Main role	Professional experience (in years)
1	EA1	Enterprise Architect	1-3
2	PM1	Project Manager	1-3
3	AD1	Agile Developer	3-6
4	AC1	Agile Coach	3-6
5	EA2	Enterprise Architect	3-6
6	EA3	Enterprise Architect	3-6
7	SA1	Solution Architect	3-6
8	PA1	Platform Architect	>6
9	PKE1	Principal Key Expert	3-6
10	AC2	Agile Coach	3-6
11	AC3	Agile Coach	>6
12	PO1	Product Owner	>6
13	EA4	Enterprise Architect	3-6
14	AD2	Agile Developer	<1

we provided a definition of each concept and presented the overall structure of the pattern language by demonstrating five examples (including their relationships to recurring concerns and other concepts) using our prototypical web application. In the following, we present the results of our evaluation[3] (see Fig. 9).

Overall evaluation of the pattern language concepts: Stakeholders were rated as the most valuable concept of the pattern language ($\mu = 1.00$; $\sigma = 0.0$). Concerns were rated the second most valuable concept ($\mu = 1.14$; $\sigma = 0.18$). The inclusion of V-Patterns was considered very valuable ($\mu = 1.43$; $\sigma = 0.41$). The usefulness of M-Patterns was received high ($\mu = 1.50$; $\sigma = 0.37$). The concept of C-Patterns was considered very useful ($\mu = 1.71$; $\sigma = 0.44$). Interviewees broadly agreed that principles should be included in the pattern language ($\mu = 1.79$; $\sigma = 0.60$). A-patterns received a positive feedback ($\mu = 2.00$; $\sigma = 0.42$). Initiatives were rated less useful ($\mu = 2.50$; $\sigma = 0.65$)

Stakeholders: The concept of stakeholders was rated indispensable (AD1), as it provides transparency about relevant roles and their concerns (EA1, EA3, PM1). One interviewee also stated that: *"patterns are developed for stakeholders"* (EA2).

Concerns: Main arguments for including concerns in the pattern language were that they provide a good starting point for decision making (EA1) and that users can directly access relevant patterns by navigating stakeholder-specific concerns (EA1, PM1). AC1 also stated that *"concerns are one of the best ways for describing stakeholder roles"*. Other respondents expressed concerns as the central motivation of the pattern language (AC1, AD2, EA2, EA3, SA1). Downsides were that concerns are too detailed (AC2), and should be made less specific (EA4). AC3 added that some concern categories such as coaching, training, human resources, etc. were missing.

V-Patterns: The interviewees mentioned the following benefits of including V-Patterns in the pattern language: they can be used as a communication medium (EA1), create a common ground between stakeholders (PO1), help with decision-making (EA4), and provide early feedback and transparency (PA1). EA2 considers V-Patterns in large-scale agile development to be as important as V-Patterns in the EAM pattern catalog, which he uses regularly [Ernst 2010; Khosroshahi et al. 2015]. AD1 and AC2 added that V-Patterns are useful as a reference book since they provide a wide-ranging toolkit for solving recurring concerns. EA3 recommended that V-Patterns should be extended by an additional section to document underlying data collection processes required for their creation.

M-Patterns: Main arguments for including M-Patterns in the pattern language were that they provide concrete guidance on how to solve problems (AC1, AD2 EA3, SA1) and that they *"represent the core of the pattern language"* (SA1). According to PO1, M-Patterns can be used as change vehicles within the organization and depend on the organizational maturity level to be applied. AD1 and EA2 claimed that the application could be difficult and may require some adaptions to be used. EA4 proposed to merge the concepts of C- and M-Patterns as *"they are very similar"*. In contrast, 71.43% of the interviewees regarded the separation of C- and M-Patterns as helpful.

C-Patterns: Interviewees stated that C-Patterns provide best practices and proven mechanisms to common coordination and communication concerns (AD2, EA2, PA1, PM1). They also show which persons have to meet and enable subject-related alignment and coordination (EA1). PKE1 added that the *variants* section of C-Patterns could be helpful to document alternative meetings formats. Although AD1 and EA3 perceived C-Patterns as very relevant in practice, they claimed that the content discussed in meetings are very complex and highly context specific, thus, making them difficult to generalize.

Principles: Positive arguments for including principles in the pattern language were that they reduce the complexity of large-scale agile development endeavors (AD2) and provide a common direction and guidance for agile teams (AD2, EA1, EA3, PO1). Interviewees highlighted that principles should be stated explicitly (AC2, PA1) as they are helpful for new employees and improve understandability (AC2). PO1 added that principles foster mindset control without prescribing concrete methodologies, thus stimulating understanding and empowerment of agile teams (P01). In this context, EA4 perceived architecture principles as essential for the success of large-scale agile development endeavors. The relationship between principles and V-Patterns was considered valuable (AC2, EA1) as the compliance of agile teams with principles can be ensured by

[3] In this paper, we do not present the evaluation results of the relationships between the pattern concepts, because their usefulness was mainly influenced by the importance the interviewees associated with the respective pattern concepts.

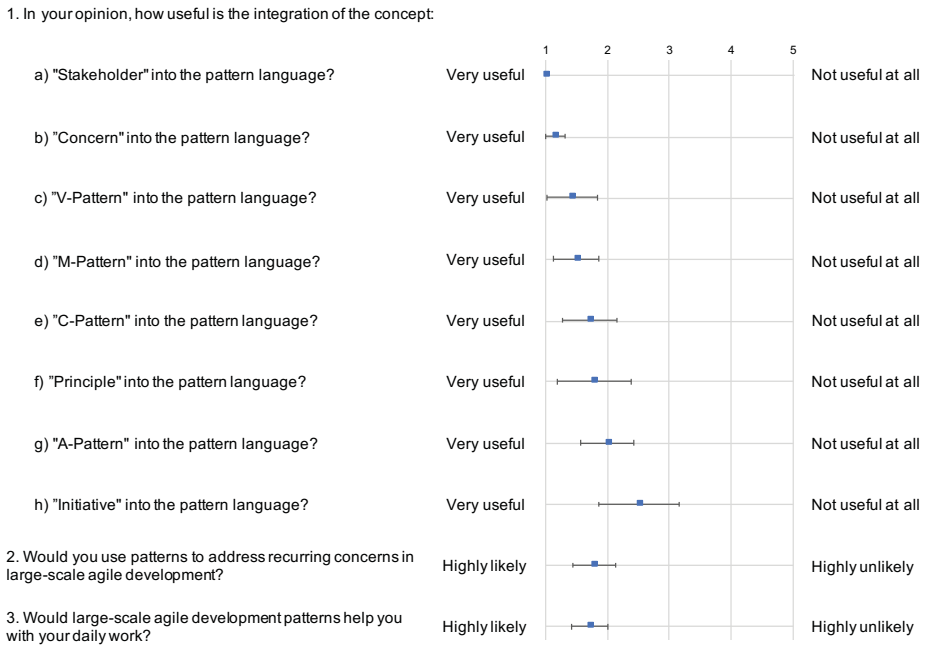


Figure 9: Evaluation results of the large-scale agile development pattern language

the use of KPIs (EA1). Although other participants acknowledged the usefulness, they rated them less applicable in this context (AD1, EA2, PM1, SA1). AC1 and EA3 had two recommendations. First, the binding nature should be included indicating mandatory or recommended principles. Second, principles should not have a *solution* section as this is the task of an M-Pattern.

A-Patterns: A-Patterns prevent pattern users from running into the same traps (EA1), show what not to do (PM1), and raise awareness of failed approaches (EA2). AD2 added that A-Patterns save time because *"you don't have to relive the same problem"*. PO1 perceived them as important but noted that *"people still make these mistakes"*. EA4 mentioned the potential disadvantage that *"everyone must understand how it is meant"*. Some interviewees questioned the usefulness of A-Patterns as they are not connected with other concepts of the pattern language (AD1, EA3, SA1). We asked whether A-Patterns should be connected with other concepts and 92.68% of interviewees replied positive. They agreed that A-Patterns should have a *problem* section, thus, being connected with concerns and stakeholders, to provide A-Patterns with more context. This enables more usability, as users can find stakeholder-relevant A-Patterns faster in comparison to other pattern languages without this connection. They also stated that the *revised solution* section should refer to C-Patterns, M-Patterns, and V-Patterns, to find appropriate solutions.

Initiatives: Some respondents said initiatives are helpful when concerns cannot be directly related to stakeholders (AC1, AD1) and if concerns occur on different organizational levels (EA1, PKE1). On the other hand, they were considered redundant (PM1) since they do not provide additional information or value (AC1, PM1, SA1). Two respondents found this concept too abstract (AC1) or not relevant for industry (EA2). In addition, they were also confused about the denotation of initiatives (AC2, AC3, PA1). Several interviewees recommended removing initiatives from the pattern language structure as it increases complexity (EA2, PKE1, PO1, SA1).

Usefulness and Support of Patterns: The majority of respondents indicated that they would use patterns to address recurring concerns in large-scale agile development.

Based on the evaluation, we made the following adjustments to the large-scale agile development pattern language structure:

- Initiatives are removed and are instead represented by the *scaling level* attribute of concerns.
- Principles are extended by a *binding nature* attribute to indicate recommended or mandatory principles.
- The *solution* section of principles is removed so that they remain generic and refer to related M-Patterns.
- A-Patterns are connected with concerns, stakeholders, C-, M-, and V-Patterns.
- V-Patterns are extended by an optional data collection attribute so that recommended data collection processes are described.

As already indicated in Section 4, the mentioned adjustments are already incorporated in the overview and meta-model of the pattern language shown in Fig. 2 and Fig. 4.

7 Conclusion and Outlook

Given the heterogeneity of large-scale projects, the nascent state of scaling agile frameworks and empirical studies [Dikert et al. 2016; Dingsøyr et al. 2019], large-scale agile development is a field susceptible to practice-driven design research. The proposed language provides the structure for documenting practice-proven solutions

to recurring large-scale agile development concerns. The pattern language includes typical stakeholders, their concerns, principles, M-Patterns, V-Patterns, C-Patterns, and A-Patterns. It also links to other standards to allow easy integration and comparison. This pattern language was evaluated by interviewing 14 large-scale agile experts from 10 organizations and by observing and documenting new patterns, four of which were presented in this paper.

The results of this paper provide compelling directions for future research. We will continue to collect data from our large-scale agile development workshops [Uludağ 2019] and case studies with industry partners. In parallel, we will conduct structured interviews with different types of stakeholders, such as agile coaches, enterprise architects or product owners, to identify role-specific concerns and pattern candidates. Based on a structured survey among companies worldwide, we will publish the *Large-Scale Agile Development Pattern Catalog* containing concerns and patterns. In future work, we will assist our industry partners to select relevant patterns and to introduce them in their organizations. This will help us to evaluate the pattern implementations in practice and to observe changing pattern implementations. With this, we also aim to close the research activity cycle of the PDR method [Buckl et al. 2013].

Acknowledgements

This work has been sponsored by the *German Federal Ministry of Education and Research (BMBF)* via the Software Campus Project SaM-IT 01IS17049 project.

References

ISO/IEC/IEEE 42010:2011(E). 2011. *Systems and software engineering – Architecture description.* Technical Report. ISO/IEC/IEEE.

Christopher Alexander. 1977. *A Pattern Language: Towns, Buildings, Construction.* Oxford University Press, New York.

Mashal Alqudah and Rozilawati Razali. 2016. A review of scaling agile methods in large software development. *International Journal on Advanced Science, Engineering and Information Technology* 6, 6 (2016), 828–837.

Scott Ambler and Mark Lines. 2012. *Disciplined agile delivery: A practitioner's guide to agile software delivery in the enterprise.* IBM Press.

Deepika Badampudi, Samuel A. Fricker, and Ana M. Moreno. 2013. Perspectives on Productivity and Delays in Large-Scale Agile Projects. In *Agile Processes in Software Engineering and Extreme Programming*, Hubert Baumeister and Barbara Weber (Eds.). Springer, Berlin, 180–194.

Mike Beedle, James O. Coplien, Jeff Sutherland, Jens C. Østergaard, Ademar Aguiar, and Ken Schwaber. 2010. Essential Scrum Patterns. In *14th European Conference on Pattern Languages of Programs.* The Hillside Group, Irsee, 1–17.

Mike Beedle, Martine Devos, Yonat Sharon, Ken Schwaber, and Jeff Sutherland. 1999. SCRUM: An Extension Pattern Language for Hyperproductive Software Development. *Pattern Languages of Program Design* 4 (1999), 637–651.

Saskia Bick, Kai Spohrer, Rashina Hoda, Alexander Scheerer, and Armin Heinzl. 2018. Coordination Challenges in Large-Scale Software Development: A Case Study of Planning Misalignment in Hybrid Settings. *IEEE Transactions on Software Engineering* 44, 10 (2018), 932–950.

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

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

Frank Buschmann, Kevlin Henney, and C. Schmidt Douglas. 2007a. *Pattern Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing.* John Wiley & Sons, Chichester.

Frank Buschmann, Kevlin Henney, and C. Schmidt Douglas. 2007b. *Pattern Oriented Software Architecture Volume 5: On Patterns and Pattern Languages.* John Wiley & Sons, Chichester.

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. 1996. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns.* John Wiley & Sons, Chichester.

James O. Coplien. 1995. A Generative Development-process Pattern Language. In *Pattern Languages of Program Design*, James O. Coplien and Douglas C. Schmidt (Eds.). ACM, New York, 183–237.

James O. Coplien. 1996. *Software Patterns: Management Briefs.* Cambridge university Press, Cambridge.

James O. Coplien and Neil B. Harrison. 2004. *Organizational Patterns of Agile Software Development.* Addison-Wesley, Boston.

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

Torgeir Dingsøyr, Davide Falessi, and Ken Power. 2019. Agile Development at Scale: The Next Frontier. *IEEE Software* (2019). Special Issue: Large-Scale Agile Development.

Torgeir Dingsøyr and Nils Moe. 2014. *Towards Principles of Large-Scale Agile Development.* Springer, Berlin, 1–8.

Amr Elssamadisy. 2008. *Agile Adoption Patterns: A Roadmap to Organizational Success.* Addison-Wesley, Boston.

Alexander M. Ernst. 2010. *A Pattern-based Approach to Enterprise Architecture Management.* Dissertation. Technische Universität München, München.

Martin Fowler. 2006. Writing Software Patterns. https://www.martinfowler.com/articles/writingPatterns.html. Accessed: 2019-02-02.

Neil B. Harrison. 1996. Organizational Patterns for Teams. In *Pattern Languages of Program Design 2*, John M. Vlissides, James O. Coplien, and Norman L. Kerth (Eds.). Addison-Wesley, Boston, 345–352.

Kevin Hoffman. 2016. *Beyond The Twelve-Factor App.* O'Reilly Media, Sebastopol.

Emam Hossain, Muhammad A. Babar, and Hye-Young Paik. 2009. Using Scrum in Global Software Development: A Systematic Literature Review. In *4th International Conference on Global Software Engineering.* IEEE, Limerick, 175–184.

Irum Inayat, Siti S. Salim, Sabrina Marczak, Maya Daneva, and Shahaboddin Shamshirband. 2015. A systematic literature review on agile requirements engineering practices and challenges. *Computers in human behavior* 51 (2015), 915–929.

Martin Kalenda, Petr Hyna, and Bruno Rossi. 2018. Scaling agile in large organizations: Practices, challenges, and success factors. *Journal of Software: Evolution and Process* 30, 10 (2018), e1954. https://doi.org/10.1002/smr.1954

Petri Kettunen. 2007. Extending Software Project Agility with new Product Development Enterprise Agility. *Software Process: Improvement and Practice* 12, 6 (2007), 541–548.

Petri Kettunen and Maarit Laanti. 2008. Combining agile software projects and large-scale organizational agility. *Software Process: Improvement and Practice* 13, 2 (2008), 183–193.

Pouya A. Khosroshahi, Matheus Hauder, Alexander W. Schneider, and Florian Matthes. 2015. *Enterprise Architecture Management Pattern Catalog Version 2.0.* Technical Report. Chair of Software Engineering for Business Information Systems (sebis), Technical University of Munich.

Henrik Kniberg and Anders Ivarsson. 2012. Scaling Agile @ Spotify.

Philippe Kruchten. 2013. Contextualizing Agile Software Development. *Journal of Software: Evolution and Process* 25, 4 (2013), 351–361.

Craig Larman and Bas Vodde. 2016. *Large-Scale Scrum: More with LeSS.* Addison-Wesley Professional.

Neil Maiden and Sara Jones. 2010. Agile Requirements Can We Have Our Cake and Eat It Too? *IEEE Software* 27, 3 (2010), 87–88.

Gerard Meszaros and Jim Doble. 1997. A Pattern Language for Pattern Writing. In *Pattern Languages of Program Design 3*, Robert C. Martin, Dirk Riehle, and Frank Buschmann (Eds.). Addison-Wesley, Boston, 529–574.

Ian Mitchell. 2016. *Agile Development in Practice.* TamaRe House, London.

Sridhar Nerur, RadhaKanta Mahapatra, and George Mangalaraj. 2005. Challenges of Migrating to Agile Methodologies. *Commun. ACM* 48, 5 (2005), 72–78.

Robert L. Nord, Ipek Ozkaya, and Philippe Kruchten. 2014. Agile in Distress: Architecture to the Rescue. In *15th International Conference on Agile Software Development.* Springer, Berlin, 43–57.

Maria Paasivaara and Casper Lassenius. 2014. Communities of practice in a large distributed agile software development organization âĂŞ Case Ericsson. *Information and Software Technology* 56, 12 (2014), 1556 – 1577. Special issue: Human Factors in Software Development.

Knut H. Rolland, Brian Fitzgerald, Torgeir Dingsøyr, and Klaas-Jan Stol. 2016. Problematizing Agile in the Large: Alternative Assumptions for Large-Scale Agile Development. In *37th International Conference on Information Systems.* Association for Information Systems, Dublin.

Scaled Agile. 2019a. PI Planning. https://www.scaledagileframework.com/pi-planning. Accessed: 2019-02-02.

Scaled Agile. 2019b. Scaled Agile Framework. https://www.scaledagileframework.com. Accessed: 2019-02-02.

Alexander W. Schneider and Florian Matthes. 2015. Evolving the EAM Pattern Language. In *20th European Conference on Pattern Languages of Programs.* ACM, New York, 45:1–45:11.

ScrumPLoP. 2019. Published Patterns. https://sites.google.com/a/scrumplop.org/published-patterns/. Accessed: 2019-02-02.

Ralph Stacey. 1996. *Strategic Management & Organizational Dynamics: The Challenge of Complexity.* Pitman Publishing, London.
Paul Taylor. 2000. Capable, productive, and satisfied: Some organizational patterns for protecting productive people. In *Pattern Languages of Program Design 4*, John M. Vlissides, James O. Coplien, and Norman L. Kerth (Eds.). Addison-Wesley, Boston, 611–636.
Ömer Uludağ. 2019. Scaling Agile Practices Workshops. https://wwwmatthes.in.tum.de/pages/1lihu1sjq8jpk/Scaling-Agile-Practices-Workshops. Accessed: 2019-02-02.
Ömer Uludağ, Martin Kleehaus, Christoph Caprano, and Florian Matthes. 2018. Identifying and Structuring Challenges in Large-Scale Agile Development Based on a Structured Literature Review. In *22nd International Enterprise Distributed Object Computing Conference*. IEEE, Stockholm, 191–197.
Antti Välimäki. 2011. *Pattern Language for Project Management in Global Software Development.* Tampere University of Technology, Tampere.
VersionOne. 2018. *12th Annual State of Agile Report.* Technical Report. VersionOne.
Etienne Wenger, Richard Arnold McDermott, and William Snyder. 2002. *Cultivating communities of practice: A guide to managing knowledge.* Harvard Business Press.
Adam Wiggins. 2017. The Twelve-Factor App. https://12factor.net/. Accessed: 2019-02-20.

A Questionnaire

A.1 General questions

Name, Organization, Role description, Personal large-scale agile development experience level, Organizational large-scale agile development experience level, Operation level, Country of headquarters, Sector, Number employees

A.2 Pattern language concepts

In your opinion, how useful is the integration of the concepts: "Stakeholder", "Initiative", "Concern", "Principle", "Coordination Pattern", "Methodology Pattern", "Viewpoint Pattern", and "Anti-Pattern" into the pattern language? (five-point Likert scale question)

Why is the concept: "Stakeholder", "Initiative", "Concern", "Principle", "Coordination Pattern", "Methodology Pattern", "Viewpoint Pattern", and "Anti-Pattern" useful / not useful for you? (open-ended question)

Does the differentiation between "Coordination Pattern" and "Methodology Pattern" help you? (yes-no question + open-ended question)

In your opinion, should the "Anti-Pattern" concept be connected to another concept from the pattern language? (yes-no question) If yes, with which concept from the pattern language should the concept "Anti-Pattern" be connected? (open-ended question)

A.3 Relationship between pattern language concepts

In your opinion, how useful is the relationship between the concepts: "Stakeholder" and "Concern", "Initiative" and "Concern", "Concern" and "Principle", "Concern" and "Coordination Pattern", "Concern" and "Methodology Pattern", "Concern" and "Viewpoint Pattern", "Principle" and "Viewpoint Pattern", "Coordination Pattern" and "Viewpoint Pattern", "Methodology Pattern" and "Viewpoint Pattern", and "Coordination Pattern" and "Methodology Pattern"? (five-point Likert scale question + open-ended question)

A.4 Discussion

Is there another concept you would like to see in the pattern language? (open-ended question)

Would you use patterns to address recurring concerns in large-scale agile development? (five-point Likert scale question)

Would large-scale agile development patterns help you with your daily work? (five-point Likert scale question)

B Current patterns and concepts in the pattern language *

B.1 Stakeholders

- **S-1:** Development Team
- **S-2:** Product Owner
- **S-3:** Scrum Master
- **S-4:** Software Architect
- **S-5:** Test Team
- **S-6:** Product Manager
- **S-7:** Program Manager
- **S-8:** Agile Coach
- **S-9:** Enterprise Architect
- **S-10:** Business Analyst
- **S-11:** Solution Architect
- **S-12:** Portfolio Manager
- **S-13:** Support Engineer
- **S-14:** UX Expert

B.2 Concerns

- **C-1:** *How to coordinate multiple agile teams that work on the same product?*
- **C-2:** *How to consider integration issues and dependencies with other subsystems and teams?*
- **C-6:** *How to manage technical debts?*
- **C-8:** *How to ensure that non-functional requirements are considered by the development team?*
- **C-9:** *How to find the right balance between architectural improvements and business value?*
- **C-13:** *How to share common vision?*
- **C-14:** *How to create a proper upfront architecture design of the system?*
- **C-21:** *How to manage dependencies to other existing environments?*
- **C-26:** *How to align and communicate architectural decisions?*
- **C-34:** *How to ensure the reuse of enterprise assets?*
- **C-41:** *How to deal with unplanned requirements and risks?*
- **C-56:** *How to define clear roles and responsibilities?*
- **C-64:** *How to define a lightweight formal review process for new technologies?*
- **C-67:** *How to encourage development teams to talk about tasks and impediments?*
- **C-78:** *How to synchronize sprints in the large-scale agile development program?*
- **C-79:** *How to ensure that the development phases are clearly separated and executed in an iterative fashion?*
- **C-80:** *How to ensure that development teams comply with architecture principles?*

B.3 A-Patterns

- **A-1:** Don't Force Traditional Project Management Concepts to Agile Software Development
- **A-2:** Don't Adopt All Agile Practices in One Go
- **A-3:** Don't Misinterpret the Meaning of Working Software Over Comprehensive Documentation
- **A-4:** Don't Spare Expenses on Agile Mindset Education
- **A-5:** Don't Assume a Tacit, Implicit Understanding of Architecture, its Scope, and the Architect's Role and Responsibility
- **A-6:** Don't Build an Ivory Tower
- **A-7:** Don't Overshoot Coordination Meetings
- **A-8:** Don't Force Traditional Job Behaviors to Agile Software Development
- **A-9:** Don't Put Individual Goals Over Team Goals
- **A-10:** Don't Creep Old Bureaucracy in Agile Software Development
- **A-11:** Don't Mix Old Approaches with Agile Software Development Approaches
- **A-12:** Don't Use Agile Practices out-of-the-box without Adapting to your Own Needs
- **A-13:** Don't Use Agile as a Golden Hammer
- **A-14:** Don't Try to Reduce the Amount of Communication in Large-Scale Agile Development Programs by Documentation
- **A-15:** Don't Consider Knowledge Sharing Strategies and Projects in Isolation
- **A-16:** Don't Add New Developers into New Teams, Instead Add Them into Existent Ones
- **A-17:** Don't Develop a Single Requirement Involved Multiple Agile Teams in Different Locations

B.4 Principles

- **P-1:** Strictly Separate Build and Run Stages

B.5 M-Patterns

- **M-1:** Cadence-Based Development
- **M-2:** Collaborative Establishment of Architecture Principles
- **M-3:** Continuous Delivery Pipeline
- **M-4:** DevOps
- **M-5:** Kanban
- **M-6:** Scrum
- **M-7:** Architectural Runway
- **M-8:** Extreme Programming
- **M-9:** ScrumXP
- **M-10:** ScrumBan
- **M-11:** Capturing NFRs in Definition of Done
- **M-12:** Architecture Spike
- **M-13:** Weighted Shortest Job First Prioritization
- **M-14:** Collaborative Adoption of new Technologies

B.6 C-Patterns

- **C-1:** Community of Practice
- **C-2:** Common Planning
- **C-3:** Scrum-of-Scrums
- **C-4:** Center of Excellence
- **C-5:** Sprint Planning
- **C-6:** Common Retrospective
- **C-7:** Sprint Review
- **C-8:** Backlog Refinement
- **C-9:** System Demo
- **C-10:** Inspect and Adapt
- **C-11:** Sprint
- **C-12:** Daily Standup
- **C-13:** Sprint Retrospective
- **C-14:** Supervision

B.7 V-Patterns

- **V-1:** Iteration Dependency Matrix
- **V-2:** Responsibility Assignment Matrix
- **V-3:** Architecture Solution Space
- **V-4:** Business Capability Map
- **V-5:** Weightest Shortest Job First
- **V-6:** Global Impediment Board
- **V-7:** SoS Board
- **V-8:** Kanban Board
- **V-9:** Portfolio Canvas
- **V-10:** SWOT Analysis
- **V-11:** ROAM Board
- **V-12:** Objectives Board
- **V-13:** Business Object Model
- **V-14:** Application Interface Map
- **V-15:** Solution Context Map
- **V-16:** Value Stream Map
- **V-17:** Persona
- **V-18:** Burndown Chart
- **V-19:** Technical Debt Backlog
- **V-20:** Speed to Market
- **V-21:** Impediment Board
- **V-22:** Starfish
- **V-23:** Run the Sail Boat

Identifying and Documenting Recurring Concerns and Best Practices of Agile Coaches and Scrum Masters in Large-Scale Agile Development

ÖMER ULUDAĞ, Technische Universität München
FLORIAN MATTHES, Technische Universität München

Ever since the release of the agile manifesto in 2001, agile methods have received widespread interest in industry and academia. Agile methods have transformed and brought unique changes to software development practices by strongly emphasizing team collaboration, change tolerance, and active customer involvement. Their proven benefits have also inspired organizations to apply them in large-scale settings. However, the adoption of agile methods at scale entails unique challenges such as coordinating and aligning multiple large-scale agile activities, dealing with internal silos, and establishing an agile culture & mindset throughout the organization. In particular, agile coaches and scrum masters are confronted with unprecedented concerns in large-scale agile development. Notwithstanding their importance for large-scale agile endeavors, extant literature still lacks an overview of their typical concerns and a collection of patterns to address them. Against this backdrop, we provide an overview of typical concerns and present five best practices of agile coaches and scrum masters in large-scale agile development.

Categories and Subject Descriptors: A.0 [**General**] Conference proceedings; K.6.3 [**Software Management**] Software Development

Additional Key Words and Phrases: agile coaches, concerns, large-scale agile development, patterns, scrum masters

ACM Reference Format:

Uludağ, Ö. and Matthes, F. 2019. Identifying and Documenting Recurring Concerns and Best Practices of Agile Coaches and Scrum Masters in Large-Scale Agile Development. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 26 (October 2019), 25 pages.

1 Introduction

Emerging in the 1990s, agile software development methods such as Extreme Programming [Beck 2000] and Scrum [Schwaber and Beedle 2001] have transformed and brought unprecedented advancements to software development practice by emphasizing change tolerance, team collaboration, and customer involvement [Kettunen 2007; Dingsøyr and Moe 2014]. With these methods, small, co-located, self-organizing teams work closely with the business customer on a single-project context, maximizing customer value and software product quality through rapid iterations and frequent feedback loops [Kettunen 2007]. Since agile methods have proved to be successful at the team level, large organizations are now aiming to scale agile methods to the enterprise level [Alqudah and Razali 2016]. Version One's 12th survey on the state of agile [VersionOne 2018] also reflects this industry trend towards adopting agile methods in-the-large. The survey shows that 52% of the 1492 respondents work in companies where the majority of teams are agile. However, the adoption of agile methods at larger scale entails

Author's address: Ö. Uludağ, Boltzmannstrasse 3, D-85748 Garching b. München; email: oemer.uludag@tum.de; F. Matthes, Boltzmannstrasse 3, D-85748 Garching b. München; email: matthes@tum.de

new challenges such as coordinating several large-scale agile activities, establishing an agile culture & mindset, and dealing with general resistances to changes [Dikert et al. 2016; Uludağ et al. 2018]. Especially agile coaches and scrum master are confronted with a number of unprecedented concerns in large-scale agile development [Uludağ et al. 2018]. Notwithstanding the significance of agile coaches and scrum masters for the success of large-scale agile endeavors, extant literature disregards an overview of their concerns and a collection of best practices to address them. Against this backdrop, we provide an overview of typical concerns of agile coaches and scrum masters and present five best practices.
The remainder of this paper is structured as follows. In Section 2, we portray the research design of our paper. In Section 3, we report on related works and describe related pattern languages. In Section 4, we elaborate the conceptual overview of the underlying pattern language to document recurring concerns and patterns. In Section 5, we give an overview of identified concerns and best practices. In Section 6, we discuss our main findings before concluding our paper with a summary of our results and remarks on future research in Section 7.

2 Research Approach

Our larger research initiative strives to document best practices that address recurring concerns of stakeholders in large-scale agile development. To balance the rigor and relevance of our research, we followed the pattern-based design research (PDR) method [Buckl et al. 2013]. The PDR method encourages researchers to theorize and learn from their intervention at industry partners while conducting rigorous and relevant design science research. The PDR method consists of four phases: *observe & conceptualize*, *pattern-based theory building & nexus instantiation*, *solution design & application*, and *evaluation & learning* (see Fig. 1).
During the *observe & conceptualize* phase, good practices for recurring concerns are observed and documented based on a typical pattern structure (see Section 4). These pattern candidates are then conceptualized by using grounding theories and evolve into genuine patterns by meeting the *rule of three*[1] [Coplien 1996], which are then integrated into the large-scale agile development pattern language. Pattern candidates, patterns, and the pattern language form an organized collection of reusable, proven solutions. In the *solution design & application* phase, stakeholders in large-scale agile development make the use of this knowledge base and select patterns based on their concerns. Selected patterns have to be configured and adjusted to the terminology of the company. After their configuration, adapted patterns can be used within the case organization. During the *evaluation & learning* phase, deviations between the actual and original pattern configuration are identified and documented. These deviations can be used to identify new best practices.

3 Related Work

Despite the industry trend towards adopting agile methods in-the-large [VersionOne 2018], sound academic research is lagging, especially regarding challenges and success factors [Dikert et al. 2016; Alsaqaf et al. 2019; Uludağ et al. 2018]. Some researchers witnessed this gap and started to publish academic papers, which are described in the following.
[Dikert et al. 2016] made a first attempt and reported 35 challenges and 29 success factors from 42 different organizations by conducting a systematic literature review of industrial large-scale agile transformations. The most salient challenge categories were *agile difficult to implement*, *integrating non-development functions*, *change resistance*, and *requirements engineering challenges*. The most important success factors were*management support*, *choosing and customizing the agile model*, *training and coaching*, and *mindset and alignment*. By means of a literature review and case study, [Kalenda et al. 2018] reported challenges and success factors of large companies adopting agile methods. They identified the following four challenges, namely *resistance to change*, *quality assurance issues*, *integrating with non-agile parts of the organization*, and *too fast roll-out*. Moreover, they

[1] The *rule of three* suggests that a documented pattern must refer to at least three known uses in practice to guarantee the re-usability of the given solution.

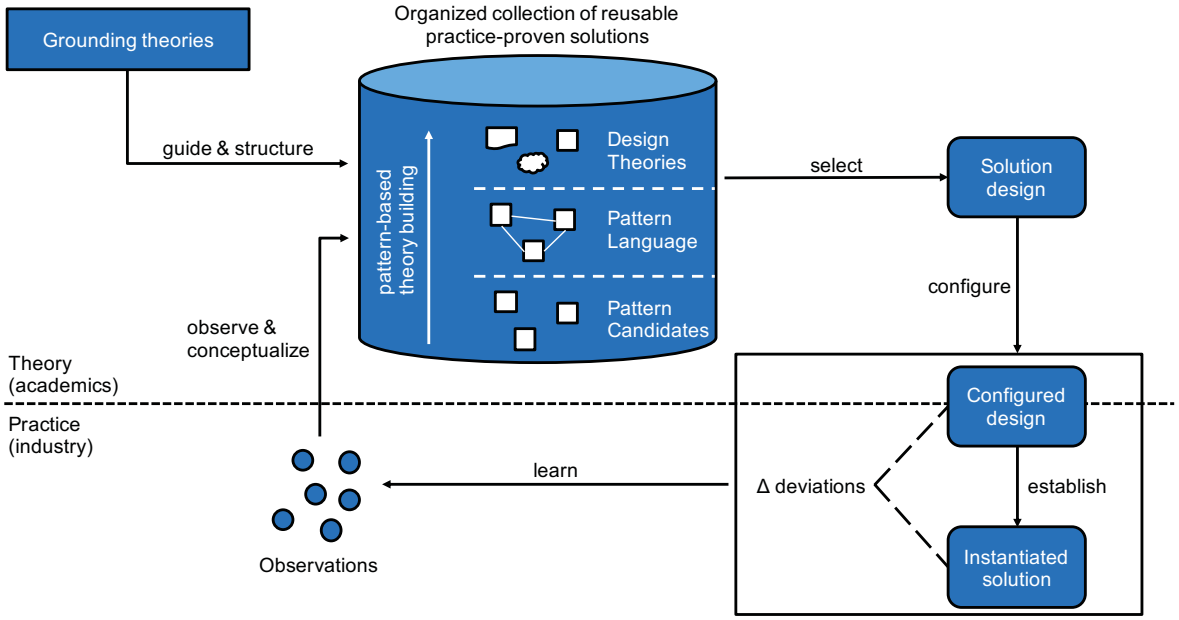


Fig. 1: Pattern-based design research [Buckl et al. 2013]

discovered four success factors, namely *unification of views and values*, *executive sponsorship and management support*, *company culture*, and *prior agile and lean experience*. In a previous study, we identified typical concerns of stakeholders and initiatives in large-scale agile development based on a structured literature review [Uludağ et al. 2018]. In total, we identified 79 concerns that were grouped into eleven challenge categories. Our previous work forms the basis for this paper, as it also included concerns and pattern candidates of agile coaches and scrum masters. [Meszaros and Doble 1997] recommend reading other related pattern languages when writing patterns. By doing that, we identified some related pattern languages as shown in Table I.

Table I. : Overview of Related Pattern Languages

Source	Scope & goal	Focus on agile development	Number of patterns	Pattern examples
[Coplien 1995]	Collection of patterns for shaping a new organization and its development processes	Partially	42	- Code Ownership - Gatekeeper - Fire Walls
[Harrison 1996]	Collection of patterns for creating effective software development teams	No	4	- Unity of Purpose - Diversity of Membership - Lock 'Em Up Together
[Beedle et al. 1999]	Collection of Scrum patterns	Yes	3	- Sprint - Backlog - Scrum Meetings
[Taylor 2000]	Collection of patterns for creating product software development environments	No	9	- Deliverables to Go - Pulse - Bootstrapping

Table I – *Continued from previous page*

Source	Scope & goal	Focus on agile development	Number of patterns	Pattern examples
[Coplien and Harrison 2004]	Collection of organizational patterns that are combined into a collection of four pattern languages	Yes	94	- Skill Mix - Demo Prep - Few Roles
[Elssamadisy 2008]	Collection of patterns for successfully adopting agile practices	Yes	38	- Refactoring - Continuous Integration - Simple Design
[Beedle et al. 2010]	Collection of the most essential best practices of Scrum	Yes	11	- Daily Scrum - Sprint Backlog - Sprint Review
[Välimäki 2011]	Enhancing performance of project management work through improved global software project management practice	Partially	18	- Collocated Kick-Off - Choose Roles in Sites - Iteration Planning
[Mitchell 2016]	Collection of patterns to address agile transformation problems	Yes	54	- Limited WIP - Kanban Sandwich - Controlled Failure
[ScrumPLoP 2019]	Body of pattern literature around agile and Scrum communities	Yes	234 (10)	- Scrum Master - Scrum of Scrums - Portfolio Standup
[Uludağ et al. 2019]	Collection of recurring concerns and patterns of typical stakeholders in large-scale agile development	Yes	70	- Strictly Separate Build and Run Stages - Community of Practice - Iteration Dependency Matrix - Don't Use Agile as a Golden Hammer

4 Large-Scale Agile Development Pattern Language

The application of agile methods on a large scale also entails unique concerns for agile coaches and scrum masters such as establishing an agile culture & mindset across the organization, facilitating coordination and communication of multiple large-scale agile endeavors, and creating information sharing and knowledge networks [Uludağ et al. 2018; Dikert et al. 2016; Alsaqaf et al. 2019; Šmite et al. 2017]. Valuable research studies providing explanations of how to address these concerns remain still scarce.

Following the idea of [Alexander 1977], the documentation of recurring concerns and best practices of agile coaches and scrum masters seems to be useful in this context. In the following, we will present the structure of our pattern language [Uludağ et al. 2019] which forms the basis for documenting of recurring concerns and patterns of agile coaches and scrum masters (see Fig. 2).

The pattern language consists of three types of patterns [Uludağ et al. 2019]:

— **Coordination Patterns (C-Patterns)** document proven coordination mechanisms to address recurring coordination concerns, i.e., managing dependencies between activities, resources or tasks.

— **Methodology Patterns (M-Patterns)** document concrete steps to be taken to address given concerns.

— **Viewpoint Patterns (V-Patterns)** document proven ways to visualize information in the form of boards, documents, metrics, models, and reports to address recurring concerns.

In addition, the pattern language comprises the following four concepts [Uludağ et al. 2019]:

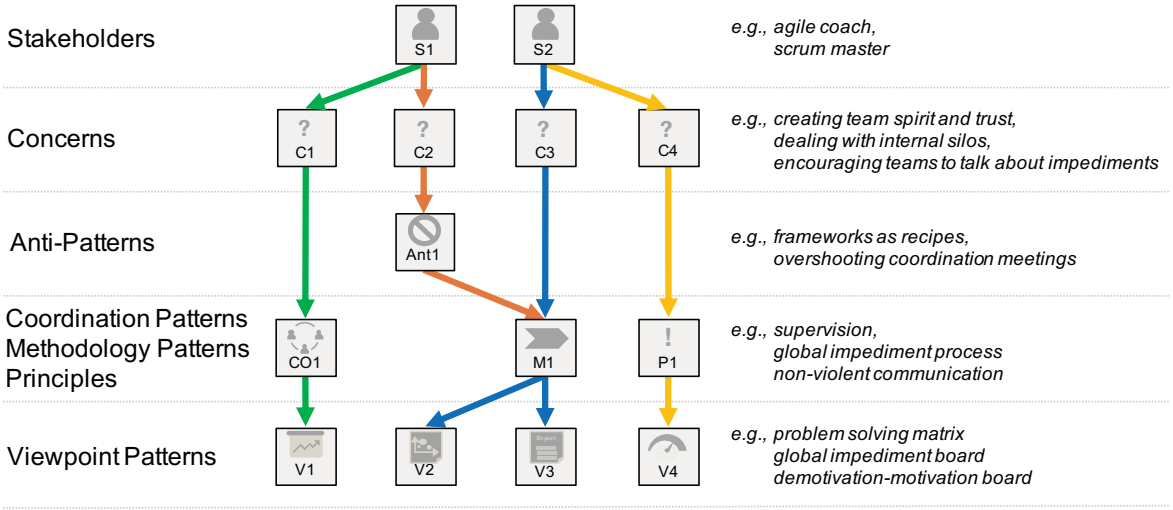


Fig. 2: Conceptual overview of the Large-Scale Agile Development Pattern Language [Uludağ et al. 2019]

— **Stakeholders** are all persons who are actively involved in, have an interest in or are in some way affected by large-scale agile development.

— **Concerns** can manifest themselves in many forms, e.g., expectations, goals, needs or responsibilities.

— **Principles** are general rules and guidelines that address given concerns by providing a common direction for action. In comparison to patterns, they do not provide any descriptions on *'how'* to address concerns.

— **Anti-Patterns (A-Patterns)** document typical mistakes and present revised solutions, which help pattern users to prevent these pitfalls.

Fig. 3 shows the attributes used to document patterns and concepts similar to those of [Buschmann et al. 1996; Ernst 2010; Coplien 1996]. All elements have an *identifier* and *name* sections to simplify referencing. Except for concerns, all elements of the pattern language have an *alias* section that contains a list of synonyms. A concern has two additional sections called *category* and *scaling level* which describe the category and the organizational level at which a concern occurs. Further, principles, patterns, and anti-patterns comprise eight common sections: the (1) *problem* and (2) *context* sections describe problems and situations to or in which they apply. The (3) *forces* section describes why the problem is tough to solve. The (4) *summary* section briefly describes the principle, pattern or anti-pattern. The (5) *consequences* section provides a list of related advantages and liabilities, while the optional (6) *other standards* and (7) *see also* sections point to other solutions and frameworks. The (8) *example* section demonstrates the problem to be addressed. Principles and patterns also have *variants* and *known uses* sections that show variants and alternatives as well as proven applications in practice. The *type* and *binding nature* sections are specific to principles and indicate their topic and whether they are recommended or mandatory. The *solution* section describes the recommended solution for a pattern. The *general form* and *revised solution* sections specific to A-Patterns delineate the not working solution and a revised solution. V-Patterns have the *type* and *data collection* sections which show the visualization concepts and collection processes necessary for their creation [Uludağ et al. 2019].
Like [Buschmann et al. 2007], we label our patterns with the star notation to denote our confidence in the maturity of a pattern. Two stars indicate that the pattern effectively solves a genuine problem in its current form. One star

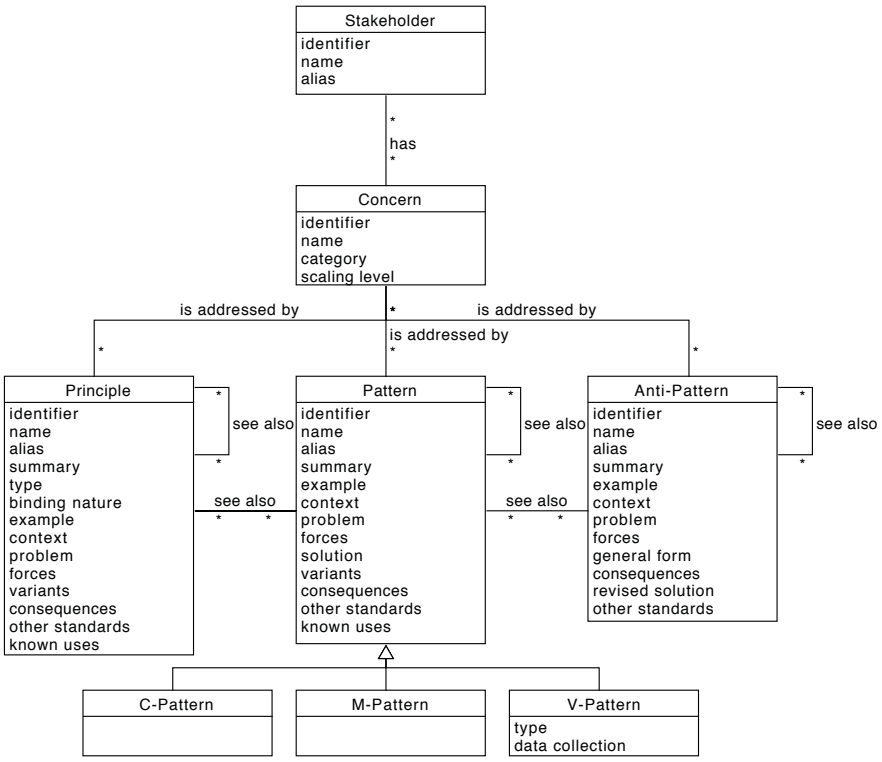


Fig. 3: Conceptual model of the Large-Scale Agile Development Pattern Language [Uludağ et al. 2019]

means that the pattern addresses a genuine problem but needs to mature. No stars denote that the pattern is a useful solution to an observed problem but requires a major revision.

5 Recurring Concerns and Best Practices

We used an integrated approach to identify recurring concerns and best practices [Cruzes and Dyba 2011]. In the first step, we created an a priori list of concerns and pattern candidates identified by a structured literature review [Uludağ et al. 2018]. In the second step, we used semi-structured interviews to prove the practical relevance of our previously identified elements as well as to extend our initial list by new concerns and best practices. All questions within the semi-structured interviews contained a combination of open and closed questions and were conversational to allow interviewees to explore their experiences and views in detail [Yin 2008]. Each interview was primarily conducted by two researchers in face-to-face meetings to facilitate observer triangulation [Runeson and Höst 2009]. A total of 13 interviews were conducted with agile coaches and scrum masters (see Table II).
In total, we observed 57 recurring concerns of agile coaches and scrum master, 36 of which were already identified by the literature review [Uludağ et al. 2018] and 21 of which were newly mentioned by the interviewees. A detailed list of identified concerns can be found in Appendix A.
We identified a total of 76 pattern candidates consisting of 21 M-Patterns, 18 V-Patterns, 14 C-Patterns, 12 Principles, and 10 A-Patterns as shown in Fig. 4. After applying the rule of three [Coplien 1996], we identified a total of 15 patterns comprising 5 M-Patterns, 2 V-Patterns, 2 C-Patterns, 4 Principles, and 2 A-Patterns. A detailed list of identified patterns can be found in Appendix B.

Table II. : Overview of interview partners

No	Role	Professional experience (in years)	Organization's experience (in years)	Industry
1	Agile Coach	3-6 years	> 6 years	IT / Technology
2	Agile Coach	3-6 years	1-3 years	Production
3	Agile Coach	1-3 years	3-6 years	Consulting
4	Agile Coach	> 6 years	1-3 years	IT / Technology
5	Agile Coach	3-6 years	< 1 year	Consulting
6	Agile Coach / Scrum Master	> 6 years	3-6 years	IT / Technology
7	Agile Coach	3-6 years	3-6 years	Consulting
8	Agile Coach	1-3 years	1-3 years	Finance / Insurance / Real Estate
9	Agile Coach	1-3 years	1-3 years	Retail
10	Agile Coach	> 6 years	1-3 years	Production
11	Agile Coach	3-6 years	1-3 years	Consulting
12	Agile Coach	3-6 years	3-6 years	Retail
13	Agile Coach	3-6 years	3-6 years	Consulting

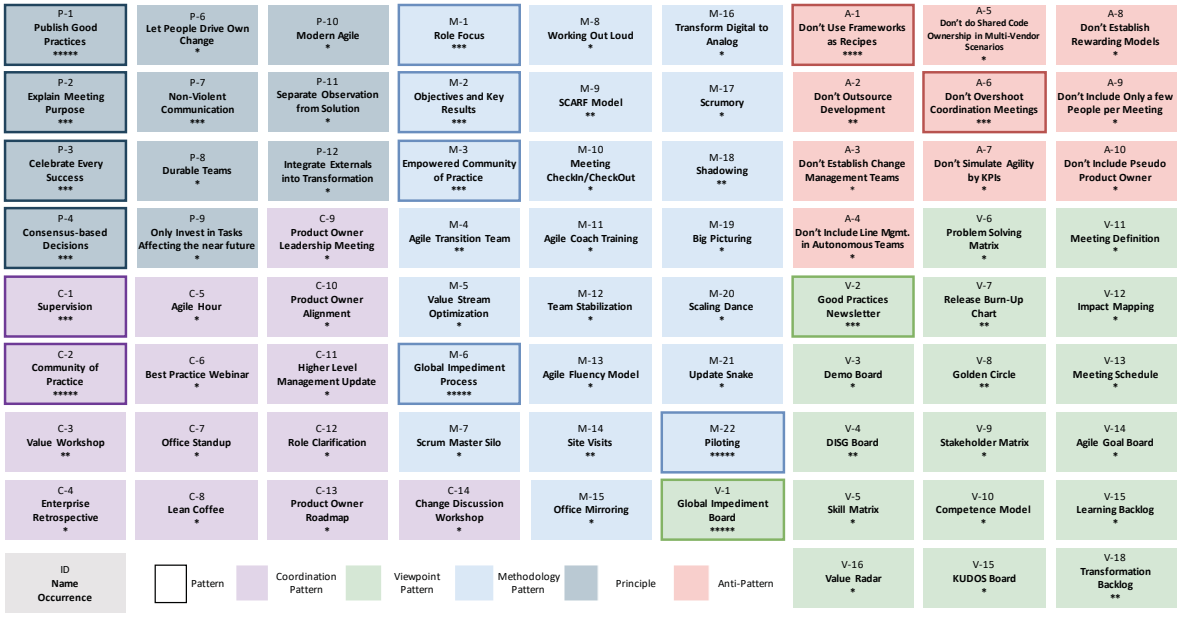


Fig. 4: Overview of identified patterns and pattern candidates

Fig. 5 depicts the current version of our pattern language, which visualizes the relationships[2] between recurring concerns and patterns of agile coaches and scrum masters. Hereafter, we present five best practices that showcase

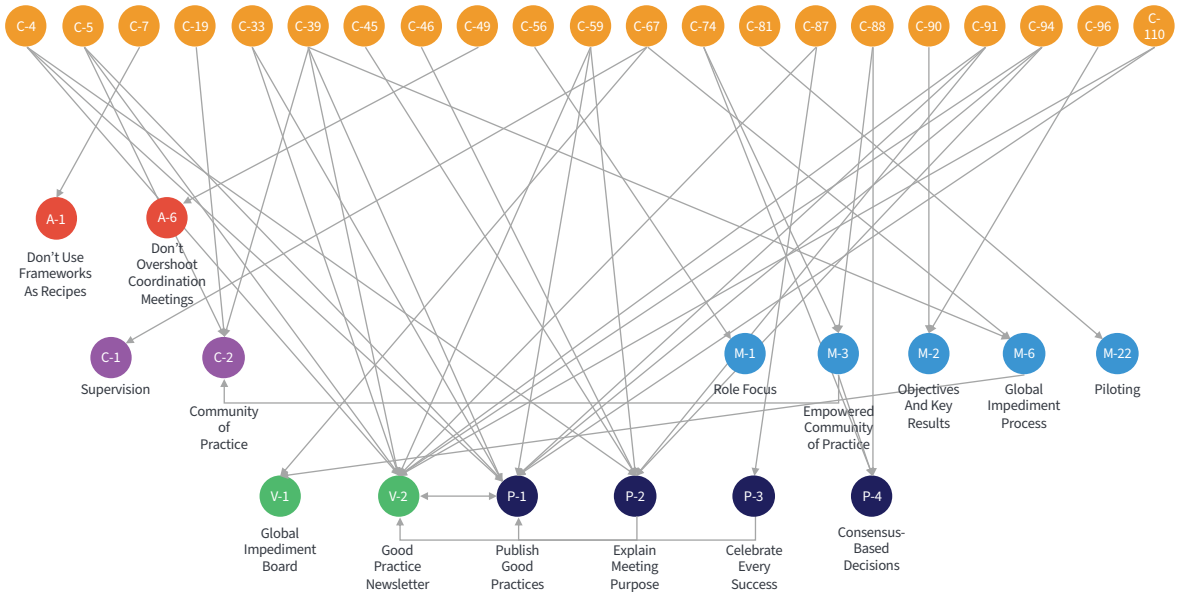


Fig. 5: Pattern language for agile coaches and scrum masters *

the different pattern types and concepts of our pattern language:

(1) **P-1:** PUBLISH GOOD PRACTICES (showing Principles),
(2) **C-1:** SUPERVISION (representing C-Patterns),
(3) **M-6:** GLOBAL IMPEDIMENT PROCESS (highlighting M-Patterns),
(4) **V-6:** GLOBAL IMPEDIMENT BOARD (illustrating (V-Patterns), and
(5) **A-1:** DON'T USE SCALING AGILE FRAMEWORKS AS A RECIPE (demonstrating A-Patterns).

[2]The arrows between the orange and other circles indicate the *is addressed by* relationship between concerns and pattern types/concepts. For instance, the concern **C-19:** *How to deal with internal silos? is addressed by* the C-Pattern **C-2:** COMMUNITY OF PRACTICE. The relationships between pattern types and/or concepts represent the *uses* or the *can be used in combination* relationship (cf. [Buschmann et al. 1996]). For example, the M-Pattern **M-2:** GLOBAL IMPEDIMENT PROCESS can be used in combination with the V-Pattern **V-1:** GLOBAL IMPEDIMENT BOARD to address the concern **C-67:** *How to encourage development teams to talk about tasks and impediments?*.

5.1 Principle: Publish Good Practices (P-1) *

Principle Overview	
Alias	
Summary	PUBLISH GOOD PRACTICES to enable a culture of open communication and continuous improvement. By applying PUBLISH GOOD PRACTICES, agile teams are encouraged to talk about things that what went well and to share their achievements with other agile teams.
Type	Agile Principle
Binding Nature	Recommended

5.1.1 *Example*

A scrum master at RetailCo successfully built a culture of kudos within his team. Although the team's motivation was highly increased by the kudos-giving culture, the scrum master did not share his success story with other scrum masters due to missing communication channels for organizational learning as well as and due to the lack of a continuous improvement culture at RetailCo.

5.1.2 *Context*

In the course of large-scale agile transformations, agile teams learn new practices and quickly achieve remarkable achievements in their new way of working. Although their achievements can also be of great importance to other agile teams, their success stories are not further communicated and the valuable knowledge remains inaccessible to the organization.

5.1.3 *Problem*

The following concerns are addressed by PUBLISH GOOD PRACTICES:

— **C-4:** *How to deal with doubts in people about changes?*
— **C-5:** *How to facilitate shared context and knowledge?*
— **C-33:** *How to build trust of stakeholders in agile practices?*
— **C-39:** *How to establish a culture of continuous improvement?*
— **C-59:** *How to establish a common understanding of agile thinking and practices?*
— **C-91:** *How to demonstrate the value add of agile methods?*
— **C-94:** *How to understand the demand for becoming agile?*
— **C-110:** *How to establish an agile mindset?*

5.1.4 *Forces*

The following forces influence PUBLISH GOOD PRACTICES:

— Agile teams are not conscious about the importance of sharing their achievements with other agile teams.
— There are no communication channels to share good practices across the organization.

5.1.5 *Consequences*

The following benefits of PUBLISH GOOD PRACTICES are known:

— A culture of continuous learning is established.
— Open communication is facilitated.
— People get informed about good practices within and outside their organization.
— People are engaged to try out new things.

The following liabilities of PUBLISH GOOD PRACTICES are known:

— Applying this principle can lead to a higher tolerance to make mistakes and trying out things that are not suitable for the organization or the team.
— An excessive use of this principle could make it difficult for agile teams to distinguish between important and unimportant good practices and to identify practices that are relevant to them.

5.1.6 *See Also*

Publish Good Practices can be used in combination with the following V-Pattern:

— **V-2:** GOOD PRACTICE NEWSLETTER

5.1.7 *Known Uses*

The following uses of PUBLISH GOOD PRACTICES are known:

— AgileConsultCo
— CarCo
— InsureCo
— ITConsultCo
— RetailCo

5.2 C-Pattern: Supervision (C-1) **

C-Pattern Overview	
Alias	
Summary	A SUPERVISION offers agile teams a platform to discuss their current problems in a small and closed circle of participants and jointly find and evaluate solutions to these problems.

5.2.1 *Example*

A scrum master at ConglomerateCo is assigned to an agile team that has an over-cautious product owner that delays the start time of the first sprint. The scrum master is overwhelmed with this situation and looks for suitable solutions to deal with this problem. At ConglomerateCo, the scrum master does not have suitable platforms to discuss his problem with other scrum master and to ask for their personal experience on similar situations.

5.2.2 *Context*

Agile teams face a variety of problems in their daily work that go beyond actual implementation challenges that are not addressed in the retrospectives for time or confidentiality reasons. Furthermore, retrospectives do not provide a suitable platform to discuss domain-specific challenges with the same agile roles.

5.2.3 *Problem*

The following concern is addressed by SUPERVISION:

— **C-67:** *How to encourage development teams to talk about tasks and impediments?*

5.2.4 *Forces*

The following forces influence SUPERVISION:

— Some employees do not like to talk openly about their problems in front of their colleagues.
— Some people do not want to raise problems with their colleagues when the people concerned are present to avoid bigger escalations.
— No suitable platforms are existing for discussing domain-specific problems with colleagues having equal roles.

5.2.5 *Solution*

Set up a SUPERVISION meeting with four to eight participants for at maximum three hours. A typical agenda of a SUPERVISION is structured as follows:

(1) **Casting**: Every participant thinks of one to two problems he wants to discuss. Every problem is shortly introduced by each participant. Afterwards, the participants vote on which of the problems are going to be discussed in the current SUPERVISION. The two most frequently chosen problems are discussed in the later part of SUPERVISION.
(2) **Telling**: The person who introduced the problem, called the storyteller, has to explain his problem in more detail. The other participants are not allowed to talk or to ask questions as long as the storyteller depicts his problem.
(3) **Asking**: At this stage, participants can ask comprehension questions to better understand the problem.
(4) **Hypothesis**: During this stage, The participants state some hypothesis on the problem. Here, the storyteller should be physically away from the other participants, e.g., by leaving the room or staying behind a flip chart, so that an intervention of the brainstorming participants is not possible. At this stage, the creativity process should not be disturbed by the storyteller.
(5) **Feedback**: The storyteller evaluates the hypotheses.
(6) **Solution**: The participants present solutions for addressing the stated problem.
(7) **Feedback**: The storyteller evaluates the proposed solutions and explains which of them are feasible and which are not.

5.2.6 *Variants*

A SUPERVISION can be done within an agile team or on a cross-team level with people from the same domain. A domain-specific SUPERVISION can focus on typical problems of agile coaches, product owners, and architects.

5.2.7 *Consequences*

The following benefits of SUPERVISION are known:

— It provides a secure environment to talk about sensitive issues.
— Based on the experiences of the collective, well-structured solutions are proposed for the problems discussed.
— Participants can reflect on the problems and solutions addressed for their own work.
— Solutions to the problems are gathered by different people, resulting in a wider range of possible solutions with each different benefits and drawbacks.

The following liabilities of SUPERVISION are known:

— Problems that are irrelevant to the participants can be neglected.
— Participants might not feel valued if their problem is not discussed.
— In the case of communicating the discussed problems with other employees outside of this circle, it can lead to a breach of trust.

5.2.8 *Known Uses*

The following uses of SUPERVISION are known:

— ConglomerateCo
— RetailCo
— SoftwareConsultCo

5.3 M-Pattern: Global Impediment Process (M-6) *

M-Pattern Overview	
Alias	
Summary	The GLOBAL IMPEDIMENT PROCESS describes a structured process for identifying, documenting, and solving impediments that affect multiple agile teams.

5.3.1 *Example*

A large-scale agile development program of RetailCo consisting of seven agile teams are about to finish their first two-week sprint. The scrum masters of these teams request access rights from the infrastructure team to use the testing environment. However, the infrastructure team is not able to process these requests since all virtual machines are already used by other teams. Consequently, the first sprint of the agile teams cannot be completed because they could not test the software sufficiently. In the respective team retrospectives, this impediment is raised by the developers. Since this is impediment is a big issue at RetailCo, the scrum masters are not able to solve it themselves. Also, RetailCo does not have a pre-defined process for escalating this impediment at the enterprise level.

5.3.2 *Context*

In agile software development, scrum masters are primarily responsible for removing impediments that may slow the development progress of their respective teams. However, in complex software development endeavors, in which multiple agile teams are involved in, these impediments are more difficult to solve as not only one agile team is affected by it but multiples. Since large-scale agile development endeavors also include other organizational units in the development process, the scrum masters do not have sufficient instruction authorities to induce other employees outside of their teams to perform specific tasks for resolving impediments of their teams.

5.3.3 *Problem*

The following concerns are addressed by GLOBAL IMPEDIMENT PROCESS:

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

— **C-67:** *How to encourage development teams to talk about tasks and impediments?*

5.3.4 *Forces*

The following forces influence GLOBAL IMPEDIMENT PROCESS:

— Impediments in large-scale agile development typically affect multiple agile teams.

— Scrum masters lack mechanisms to escalate larger impediments to higher organizational levels so that these are resolved by middle management or even by the executive board.

— Scrum masters do not have sufficient instruction authorities of employees outside of their teams that should take actions in order to resolve the impediments.

— Scrum masters have difficulties in identifying relevant employees outside of their teams that should own and resolve the impediments.

5.3.5 *Solution*

Implement a GLOBAL IMPEDIMENT PROCESS to tackle impediments that scrum masters cannot solve on their own. Each impediment of a team is included within the GLOBAL IMPEDIMENT PROCESS if the team cannot solve the impediment by its own. The process is structured according to Fig. 6.

The process includes a *Global Impediment Working Group* that consists of people having an overarching view of the organization, therefore knowing the right people to solve a global impediment. The Global Impediment Working Group meets every two weeks and discusses and prioritizes new impediments. They add them to the

Global Impediment Board and try to solve the impediments. Because of their knowledge about the company, the probability that they know people who can solve the impediment is very high.

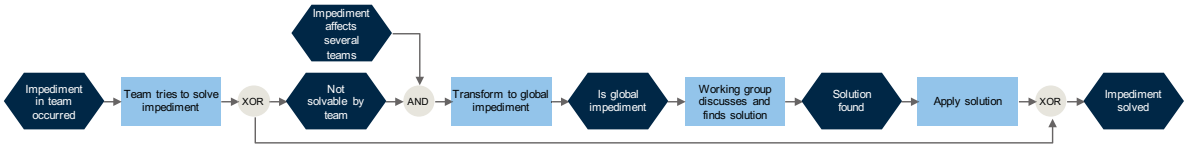


Fig. 6: Event-Driven GLOBAL IMPEDIMENT PROCESS

5.3.6 *Variants*

The following variants are known for the GLOBAL IMPEDIMENT PROCESS:

— Each impediment has an 'owner', who is someone from the Working Group who also knows about the difficulty of the impediment.
— Each impediment has a 'supporter', who is someone who as a major influence on the solution of the impediment.
— An impediment can be documented by using the A3 format [Sobek and Jimmerson 2004].

5.3.7 *Consequences*

The following benefits of GLOBAL IMPEDIMENT PROCESS are known:

— Impediments get solved.
— Prioritization enables calculation of real cost caused by an impediment. This may increase resolution speed.
— The process enables transparency.
— The process minimized local applications and workarounds.
— By resolved impediments, the velocity of agile teams is not hampered.

The following liabilities of GLOBAL IMPEDIMENT PROCESS are known:

— The process requires increased effort for the participants of the Global Impediment Working Group.

5.3.8 *See Also*

The GLOBAL IMPEDIMENT PROCESS uses the following V-Pattern:

— **V-1:** GLOBAL IMPEDIMENT BOARD

5.3.9 *Known Uses*

The following uses of GLOBAL IMPEDIMENT PROCESS are known:

— AutonomousDrivingCo
— ConglomerateCo
— RetailCo
— SoftwareCo
— SoftwareConsultCo

5.4 V-Pattern: Global Impediment Board (V-6)*

V-Pattern Overview	
Alias	Global Impediment Backlog
Summary	A GLOBAL IMPEDIMENT BOARD shows all impediments of an organization which are ether not solvable by an agile team itself or are relevant for several teams in a company.
Type	Board

5.4.1 *Example*

RetailCo has established a GLOBAL IMPEDIMENT PROCESS to handle impediments that scrum masters cannot solve on their own or are relevant for multiple agile teams. Although RetailCo has established a Global Impediment Working Group for resolving these type of impediments, it neither uses a structured format for documenting global impediments nor stores them in a central database. In addition, the Global Impediment Working Group does not prioritize global impediments. Thus, it does not directly addressed urgent impediments that can have a significant impact on agile teams.

5.4.2 *Context*

The organization has already implemented the M-Pattern GLOBAL IMPEDIMENT PROCESS. The resolution of impediments is neither documented in a uniform format nor stored centrally so that employees have difficulties in identifying current or historical global impediments. In addition, the organization does not know which of the impediments are important or urgent to resolve.

5.4.3 *Problem*

The following concern is addressed by GLOBAL IMPEDIMENT BOARD:

— **C-67:** *How to encourage development teams to talk about tasks and impediments?*

5.4.4 *Forces*

The following forces influence GLOBAL IMPEDIMENT BOARD:

— Global Impediments need to be centrally managed and tracked so that they are actually resolved by the impediment owners.

— Different employees may have different styles of documenting impediments that would make it difficult to compare impediments.

— Due to numerous tools in the area of large-scale agile development, it can be difficult to find global impediments in the right place.

— Due to vast numbers of global impediments, it can be difficult to distinguish important/urgent impediments from unimportant/non-urgent impediments.

5.4.5 *Solution*

Set up a GLOBAL IMPEDIMENT BOARD to manage all global impediments throughout the GLOBAL IMPEDIMENT PROCESS. In large-scale agile development, a list with the following structure is frequently used:
The ID is a consecutive, unique integer value that is used to identify an impediment. The prioritization is done by the Global Impediment Working Group and indicates the urgency of the impediment. Impediments with higher prioritization should be solved first. 'Handed in by' refers to the team or individual who handed in the impediment. The owner is someone from the Working Group, who is responsible for solving the impediment. The 'A3'-attribute is optional if the GLOBAL IMPEDIMENT PROCESS requires the submission of an impediment in the A3 format.
The underlying information model of the GLOBAL IMPEDIMENT BOARD can be found in Fig. 8.

Global Impediment Board

ID	Prioritization	Name	Handed in by	Description	Date	Owner	A3	Status
...	...	...	...	...	...	...	...	...
100	3	Dev Access	Team A	Customer cannot access development environment due to security guidelines	06/05/2019	John Doe	Link to A3	Ongoing
...	...	...	...	...	...	...	...	...

Fig. 7: Exemplary view for GLOBAL IMPEDIMENT BOARD

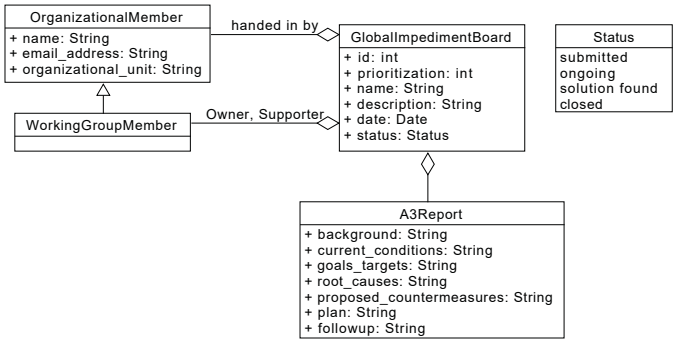


Fig. 8: Underlying information model of GLOBAL IMPEDIMENT BOARD

5.4.6 *Variants*

Depending on the organization, different attributes can be added or removed.

5.4.7 *Consequences*

The following benefits of GLOBAL IMPEDIMENT BOARD are known:

— Everyone within the organization can view current global impediments and see if anyone else has a similar problem.

— Prioritization enables faster solving of emerging impediments that have a large impact on a team's organization.

— Global impediments can be compared more easily with each other.

The following liabilities of GLOBAL IMPEDIMENT BOARD are known:

— It requires effort to manage the GLOBAL IMPEDIMENT BOARD.

5.4.8 *Data Collection*

The GLOBAL IMPEDIMENT BOARD can be digitally administrated in any digital collaboration tool by the Global Impediment Working Group. It is updated whenever an impediment occurs by a member of the Working Group. Only these members have writing permissions. It depends on the organization if the board should be public or kept private to the Working Group.

5.4.9 *See Also*

GLOBAL IMPEDIMENT BOARD is used by the following M-Pattern:

— **M-8:** GLOBAL IMPEDIMENT PROCESS

5.4.10 *Known Uses*

The following uses of GLOBAL IMPEDIMENT BOARD are known:

— AutonomousDrivingCo
— ConglomerateCo
— RetailCo
— SoftwareCo
— SoftwareConsultCo

5.5 A-Pattern: Don't Use Scaling Agile Frameworks as a Recipe (A-1) *

A-Pattern Overview	
Alias	Scaling Agile Frameworks Aren't Silver Bullets
Summary	The A-Pattern DON'T USE SCALING AGILE FRAMEWORKS AS A RECIPE shows why it is not advisable to over rely on scaling agile frameworks without training people in agile values and principles and tailoring these frameworks to the specific needs of the organization.

5.5.1 *Example*

The executive board of RetailCo decides to revive a failed customer relationship project by using agile methods. Due to the complexity of the project, the management decide to relaunch it with the help of a scaling framework. After a short internet research and discussion with external agile coaches, the management decides to relaunch the project by adopting the Scaled Agile Framework[3] (SAFe). In order not to fail, the management decides to use the most comprehensive configuration of the SAFe framework, namely the Full SAFe configuration. After hiring several external agile coaches and offering one-day SAFe training courses for the employees, the large-scale agile development program consisting with three agile teams starts with the first program increment. The management of RetailCo stipulated the large-scale agile development program that all practices, artefacts and roles specified by Full SAFe are to be applied. After a few program increments the large-scale agile development program notices that the used framework is causing some problems such as additional workload or complexity. Consequently, the large-scale agile development program compulsively tries to solve the problems arising from the use of Full SAFe and neglects to address the actual problems of the project.

5.5.2 *Context*

As today's competitive environments become increasingly turbulent and the software systems to be developed become more complex, traditional companies increasingly decide to adopt scaling agile frameworks for their software development endeavors. Thus, several scaling agile frameworks, such as DAD[4], LeSS[5], and SAFe[6] were proposed by practitioners to resolve issues associated with team size, customer involvement, and project constraints that are mostly applied by traditional organizations.

5.5.3 *Problem*

The following concern is addressed by DON'T USE SCALING AGILE FRAMEWORKS AS A RECIPE:

— **C-7:** *How to deal with incorrect practices of agile development?*

5.5.4 *Forces*

The following forces occur in the context of DON'T USE SCALING AGILE FRAMEWORKS AS A RECIPE:

— The vendors of scaling agile frameworks promise organizations that the full potential of scaling agile frameworks will only be achieved when companies implement the frameworks exactly as they require.

— Companies are tempted by the fact that the frameworks have already been successfully implemented in other organizations and therefore automatically assume that these will also solve current problems in their organization.

— Due to the lack of an agile mindset, companies believe that the mere use of scaling agile frameworks is sufficient to realize the benefits of agile development, neglecting the importance of agile principles and values.

[3] https://www.scaledagileframework.com/
[4] https://disciplinedagiledelivery.com/
[5] https://less.works/
[6] https://www.scaledagileframework.com/

— Due to their previous way of thinking, traditional organizations tend to think in terms of predefined templates and processes, thus limiting the ability to discover new ideas and ways of working.

5.5.5 *General Form*

Traditional organizations that worked for a long period in hierarchical structures and plan-driven software development methods tend to over-rely on predefined structures, processes, and rules. However, this mindset is also exercised when organizations decide to adopt scaling agile frameworks as a basis for their complex software development endeavors. Thus, these type of organizations adopt the practices, artefacts, and roles proposed by the frameworks without to question whether they are useful for the addressing the actual problems of the organization. In this context, organizations focus more on the appropriate implementation of the adopted scaling agile framework than on understanding the values and principles behind the framework. This phenomenon is also known as *'method prison'*.

5.5.6 *Consequences*

The following benefits of DON'T USE SCALING AGILE FRAMEWORKS AS A RECIPE are known:

— Scaling agile frameworks provide detailed guidance for applying agile practices.
— Scaling agile frameworks constitute an entry point for hierarchical organizations to establish an agile culture across the company.
— The usage of scaling agile frameworks increases the productivity of the organization.
— Scaling agile frameworks provide quick answers for typical software process problems.

The following liabilities of DON'T USE SCALING AGILE FRAMEWORKS AS A RECIPE are known:

— By relying too much on scaling agile frameworks, employees are not encouraged to understand the values and principles behind these frameworks and do not develop an agile mindset.
— Not all practices, roles, and artefacts of scaling agile frameworks apply to an enterprise, so wasting time, money, and effort is put into their application.
— Since scaling agile frameworks represent a simplified representation of reality, they are not designed to address more complex problems from reality.

5.5.7 *Revised Solution*

Don't adopt an agile framework one-to-one. Always analyze which practices are relevant to the organization and which are not. Start a small-scale pilot first, and scale it to the whole organization after a successful pilot. Constantly inspect the organization and react to inefficiency accordingly.
Additionally, teach the organization to not only apply agile methods but to act and work according to agile values and principles. This requires extensive training and continuous review and improvement. Values and principles are the basis for an efficient and value-creating organization.

5.5.8 *See Also*

The A-Pattern DON'T USE SCALING AGILE FRAMEWORKS AS A RECIPE can be avoided by using the following principle:

— **P-17:** SEPARATE OBSERVATION FROM SOLUTION

6 Discussion

In the following, we discuss the main findings of our study.

(1) Adaptation of scaling agile frameworks to company contexts
The majority of the interviewed organizations adopted a variety of scaling agile frameworks for supporting their product development such as Large-Scale Scrum[7], Scaled Agile Framework[8], Scrum of Scrums[9], and Scrum at Scale[10]. Some of the interviewed companies invented their frameworks mainly for scaling agile practices over multiple teams. In larger companies, we observed that various scaling agile frameworks were used in different product development units. Furthermore, we noticed that the importance of the frameworks for the companies differed significantly. While in some organizations the correct implementation of a certain framework was regarded as very important for the success of the product development, other companies considered scaling agile frameworks as a means to an end. Typically, the latter type of organizations used the frameworks as inspiration for their product development. Organizations concentrating too much on the proper implementation of a specific framework tended to fall into the pitfall of using frameworks as recipes (see A-Pattern DON'T USE SCALING AGILE FRAMEWORKS AS A RECIPE). Although many framework vendors strongly recommend the complete and unmodified use of their frameworks, the interviewees mentioned that their organizations have not adopted the scaling of agile frameworks one-to-one, but have tailored them to their context and needs. These adaptations included: (1) renaming of roles, practices, and artifacts, (2) introducing new roles based on the current organizational unit, (3) initiating new coordination meetings due to increased coordination needs, and (4) omitting recommended roles, events, and artifacts in order not to further increase the complexity of product development.

(2) Risk of patterns being used as cooking recipes
Our decision to document best practices in the form of patterns received positive feedback by the agile and scrum masters as well as by the interviewees from our previous study (cf. [Uludağ et al. 2019]). On the one hand, the interviewees asked themselves how these patterns could be used pedagogically to train new employees in the field of large-scale agile development. On the other hand, other participants considered using the patterns for upcoming projects. Similar to scaling agile frameworks, the usage of patterns also harbors some risks that must be mentioned here. First, patterns are context-specific signifying that some patterns may work very well in some companies, while the same patterns may not be suitable for other organizations. For instance, while some organizations preferred to introduce change teams (see M-Pattern AGILE TRANSITION TEAM) aiming to lead the agile transformations within the organizations, others stated that establishing change teams might not work, and thus should be avoided (see A-Pattern DON'T ESTABLISH CHANGE MANAGEMENT TEAMS). One way to mitigate this risk is by applying the PDR method presented in Section 2. According to the PDR method, a researcher should support organizations in the selection of suitable patterns and their configuration for the organizational context. In addition, the researcher should document possible deviations and, if necessary, revise the initial pattern, e.g., by updating the consequences or variants sections of a pattern. Second, similar to the application of scaling agile frameworks, employees may focus excessively on the correct application of a pattern rather than understanding the actual problem and intentions behind the pattern. Thus, we highly recommend that patterns should be used as decision-support and should not anticipate decisions.

(3) Agile values and principles vs. agile practices
During the interviews, many agile coaches and scrum master pointed out that many employees wrongly equate

[7] https://less.works/less/framework/index.html
[8] https://www.scaledagileframework.com/
[9] https://www.scruminc.com/scrum-of-scrums/
[10] https://www.scrumatscale.com/scrum-at-scale-guide/

agility by using agile practices without understanding the underlying values and principles, e.g., moving cards on Kanban boards or replacing *'old and boring'* software development terminologies with *'new and fancy'* agile development terminologies without knowing *'why'* these practices are important. As a consequence, many companies become *'pseudo agile'* which increasingly encounter cultural problems, since the agile mindset of the employees is still very immature. Also, the interviewed agile coaches and scrum masters indicated that learning new practices is much easier than understanding the underlying values and principles. They explained this problem by using the metaphor of an iceberg, i.e., the visible part of an iceberg represents agile practices and the important part of the iceberg, which is underwater, is made of agile principles and values. Influencing values and principles which are not visible is more difficult than influencing the visible practices.

(4) Process-oriented agile coaches vs. mindset-oriented agile coaches
We observed two types of agile coaches in our interviews, namely *process-oriented* and *mindset-oriented* agile coaches. The process-focused coaches were mainly concerned about the proper application of agile practices and methods by the teams. To this end, they mainly proposed best practices in the form of M-Patterns such as OBJECTIVES AND KEY RESULTS and GLOBAL IMPEDIMENT PROCESS. On the other hand, the mindset-focused coaches often explained concerns about the adoption of an agile mindset across the organization or the creation of an ideal working environment for agile teams to foster team communication and collaboration. The mindset-oriented coaches named typically principles, workshops, and visualizations for resolving their concerns, such as the Principle PUBLISH GOOD PRACTICES and the C-Pattern SUPERVISION. These observations are consistent with the concept of [Kelly 2008], according to which there are two different approaches to coaching: *directive* and *non-directive* coaching. The process-focused coaches often applied a directive approach, while the mindset-focused coaches mostly used a non-directive coaching approach. With directive coaching, the agile coach has extensive knowledge of the domain and mostly trains a team in the application of agile practices. Thus, directive coaching can be used to adopt agile practices and help teams to work in new ways. In contrast, the non-directive approach does not necessarily require the coach to be an expert in the field. Instead, the coach tries to help the teams focus on their own goals and work towards achieving them. This form of coaching helps the teams to grow on their own and to improve their performance. Non-directive coaching is more suitable for teams that are already familiar with agile practices, while the directive approach is more suited for new teams [Kelly 2008].

7 Conclusion and Outlook

The success of agile methods for small teams has inspired large enterprises to apply them on a larger scale to build complex software systems [Dingsøyr and Moe 2014; Alqudah and Razali 2016]. The scaling of agile methods entails key managerial challenges such as coordinating multiple large-scale agile endeavors, establishing an agile mindset across the organization, and facing general resistances to changes [Uludağ et al. 2018; Dikert et al. 2016; Alsaqaf et al. 2019]. Especially agile coaches and scrum master are confronted with a number of unprecedented concerns in large-scale agile development [Uludağ et al. 2018]. Notwithstanding the significance of agile coaches and scrum masters for the success of large-scale agile endeavors, extant literature disregards an overview of their concerns and a collection of best practices to address them. Against this backdrop, we interviewed 13 agile coaches and scrum masters and identified 57 recurring concerns and 15 best practices, five of which were presented in this paper.
Finally, this paper leaves some room for future research. First, we aim to conduct more interviews with other typical stakeholders in large-scale agile development, such as product owners, solution architects, and developers. These interviews will help us to identify new role-specific concerns, patterns, and pattern candidates. Second, by means of a structured questionnaire among companies worldwide, we will publish the Large-Scale Agile Development Pattern Catalog containing concerns and patterns. Third, we will assist agile coaches and scrum masters of our industry partners in selecting relevant patterns and introducing them into their organizations. This will allow us to

observe actual pattern instantiations and to identify possible deviations from the originally introduced patterns. Thereby, we also intend to close the research activity cycle of the PDR method [Buckl et al. 2013].

Acknowledgements

This work has been sponsored by the *German Federal Ministry of Education and Research (BMBF)* via the Software Campus Project SaM-IT 01IS17049 project.

REFERENCES

Christopher Alexander. 1977. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, New York.

Mashal Alqudah and Rozilawati Razali. 2016. A review of scaling agile methods in large software development. *International Journal on Advanced Science, Engineering and Information Technology* 6, 6 (2016), 828–837.

Wasim Alsaqaf, Maya Daneva, and Roel Wieringa. 2019. Quality requirements challenges in the context of large-scale distributed agile: An empirical study. *Information and Software Technology* 110 (2019), 39 – 55.

Kent Beck. 2000. *Extreme programming explained: embrace change*. Addison-Wesley.

Mike Beedle, James O. Coplien, Jeff Sutherland, Jens C. Østergaard, Ademar Aguiar, and Ken Schwaber. 2010. Essential Scrum Patterns. In *14th European Conference on Pattern Languages of Programs*. The Hillside Group, Irsee, 1–17.

Mike Beedle, Martine Devos, Yonat Sharon, Ken Schwaber, and Jeff Sutherland. 1999. SCRUM: An Extension Pattern Language for Hyperproductive Software Development. *Pattern Languages of Program Design* 4 (1999), 637–651.

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

Frank Buschmann, Kevlin Henney, and C. Schmidt Douglas. 2007. *Pattern Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing*. John Wiley & Sons, Chichester.

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. 1996. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. John Wiley & Sons, Chichester.

James O. Coplien. 1995. A Generative Development-process Pattern Language. In *Pattern Languages of Program Design*, James O. Coplien and Douglas C. Schmidt (Eds.). ACM, New York, 183–237.

James O. Coplien. 1996. *Software Patterns: Management Briefs*. Cambridge university Press, Cambridge.

James O. Coplien and Neil B. Harrison. 2004. *Organizational Patterns of Agile Software Development*. Addison-Wesley, Boston.

Daniela S Cruzes and Tore Dyba. 2011. Recommended steps for thematic synthesis in software engineering. In *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on*. IEEE, 275–284.

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

Torgeir Dingsøyr and Nils Moe. 2014. *Towards Principles of Large-Scale Agile Development*. Springer, Berlin, 1–8.

Amr Elssamadisy. 2008. *Agile Adoption Patterns: A Roadmap to Organizational Success*. Addison-Wesley, Boston.

Alexander M. Ernst. 2010. *A Pattern-based Approach to Enterprise Architecture Management*. Dissertation. Technische Universität München, München.

Neil B. Harrison. 1996. Organizational Patterns for Teams. In *Pattern Languages of Program Design 2*, John M. Vlissides, James O. Coplien, and Norman L. Kerth (Eds.). Addison-Wesley, Boston, 345–352.

Martin Kalenda, Petr Hyna, and Bruno Rossi. 2018. Scaling agile in large organizations: Practices, challenges, and success factors. *Journal of Software: Evolution and Process* 30, 10 (2018), e1954. DOI:https://doi.org/10.1002/smr.1954

Allan Kelly. 2008. *Changing software development: Learning to become agile*. John Wiley & Sons.

Petri Kettunen. 2007. Extending Software Project Agility with new Product Development Enterprise Agility. *Software Process: Improvement and Practice* 12, 6 (2007), 541–548.

Gerard Meszaros and Jim Doble. 1997. A Pattern Language for Pattern Writing. In *Pattern Languages of Program Design 3*, Robert C. Martin, Dirk Riehle, and Frank Buschmann (Eds.). Addison-Wesley, Boston, 529–574.

Ian Mitchell. 2016. *Agile Development in Practice*. TamaRe House, London.

Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering* 14, 2 (2009), 131.

Ken Schwaber and Mike Beedle. 2001. *Agile Software Development with Scrum* (1st ed.). Prentice Hall, Upper Saddle River.

ScrumPLoP. 2019. Published Patterns. https://sites.google.com/a/scrumplop.org/published-patterns/. (2019). Accessed: 2019-02-02.

Darja Šmite, Nils Brede Moe, Aivars Šāblis, and Claes Wohlin. 2017. Software teams and their knowledge networks in large-scale software development. *Information and Software Technology* 86 (2017), 71–86.

Durward K. Sobek and Cindy Leduc Jimmerson. 2004. A 3 Reports : Tool for Organizational Transformation. *Industrial Engineering Research Conference*.

Paul Taylor. 2000. Capable, productive, and satisfied: Some organizational patterns for protecting productive people. In *Pattern Languages of Program Design 4*, John M. Vlissides, James O. Coplien, and Norman L. Kerth (Eds.). Addison-Wesley, Boston, 611–636.

Ömer Uludağ, Nina Harders, and Florian Matthes. 2019. Documenting Recurring Concerns and Patterns in Large-Scale Agile Development. In *24th European Conference on Pattern Languages of Programs*. ACM, New York.

Ömer Uludağ, Martin Kleehaus, Christoph Caprano, and Florian Matthes. 2018. Identifying and Structuring Challenges in Large-Scale Agile Development Based on a Structured Literature Review. In *22nd International Enterprise Distributed Object Computing Conference*. IEEE, Stockholm, 191–197.

Antti Välimäki. 2011. *Pattern Language for Project Management in Global Software Development*. Tampere University of Technology, Tampere.

VersionOne. 2018. *12th Annual State of Agile Report*. Technical Report. VersionOne.

Robert K. Yin. 2008. *Case Study Research: Design and Methods*. Sage Publications, London.

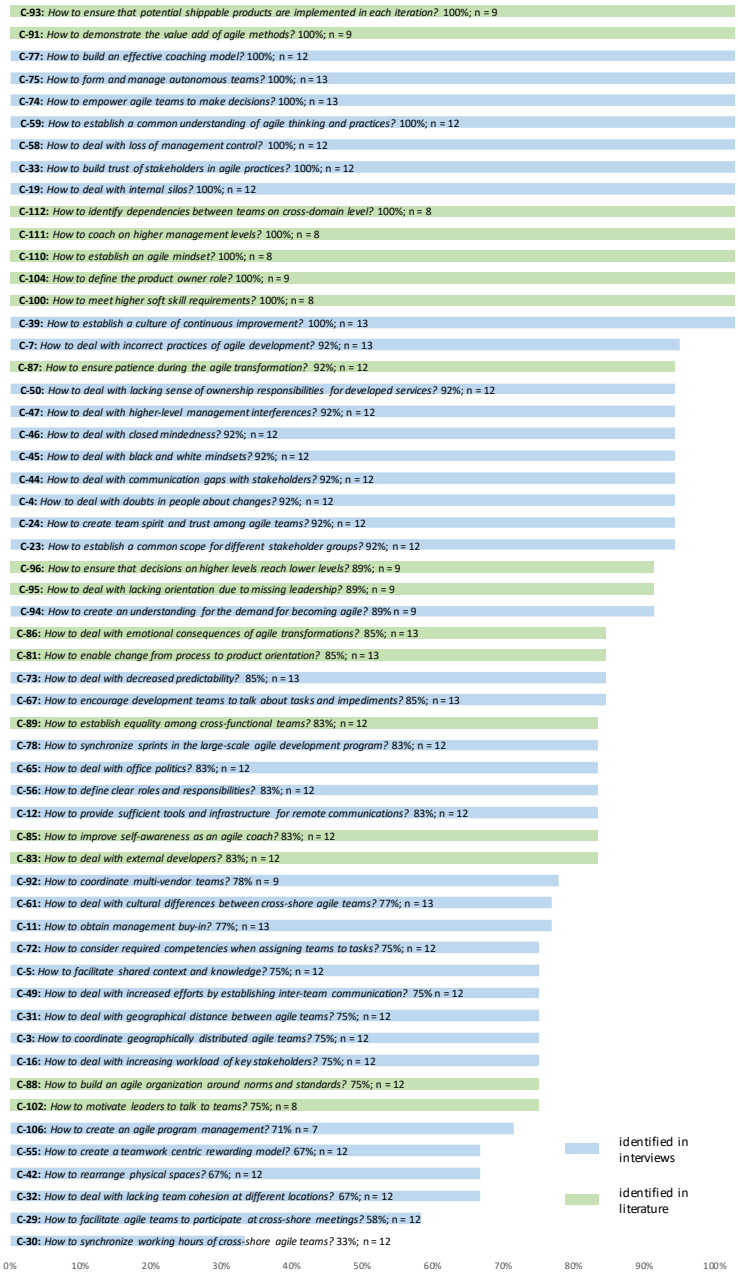


Fig. 9: Overview of identified concerns of agile coaches and scrum masters

B Pattern Language for Agile Coaches and Scrum Masters *

B.1 Concerns

— **C-4:** *How to deal with doubts in people about changes?*
— **C-5:** *How to facilitate shared context and knowledge?*
— **C-19:** *How to deal with internal silos?*
— **C-33:** *How to build trust of stakeholders in agile practices?*
— **C-39:** *How to establish a culture of continuous improvement?*
— **C-46:** *How to deal with closed mindedness?*
— **C-49:** *How to deal with increased efforts by establishing inter-team communication?*
— **C-56:** *How to define clear roles and responsibilities?*
— **C-59:** *How to establish a common understanding of agile thinking and practices?*
— **C-67:** *How to encourage development teams to talk about tasks and impediments?*
— **C-74:** *How to empower agile teams to make decisions?*
— **C-81:** *How to enable the change from process to product orientation?*
— **C-87:** *How to ensure patience during the agile transformation?*
— **C-88:** *How to build an agile organization around norms and standards?*
— **C-90:** *How to deal with the lack of objective measuring methods?*
— **C-91:** *How to demonstrate the value add of agile methods?*
— **C-94:** *How to understand the demand for becoming agile?*
— **C-96:** *How to ensure that decisions on higher levels reach lower levels?*
— **C-110:** *How to establish an agile mindset?*

B.2 A-Patterns

— **A-1:** Don't Use Frameworks as Recipes
— **A-6:** Don't Overshoot Coordination Meetings

B.3 Principles

— **P-1:** Publish Good Practices
— **P-2:** Explain Meeting Purpose
— **P-3:** Celebrate Every Success
— **P-4:** Consensus-Based Decisions

B.4 M-Patterns

— **M-1:** Role Focus
— **M-2:** Objectives and Key Results
— **M-3:** Empowered Community of Practice
— **M-6:** Global Impediment Process
— **M-22:** Piloting

B.5 C-Patterns

— **C-1:** Supervision
— **C-2:** Community of Practice

B.6 V-Patterns

— **V-1:** Global Impediment Board
— **V-2:** Good Practice Newsletter

PLoP'19, OCTOBER 7–10, Ottawa, Ontario, Canada. HILLSIDE 978-1-941652-03-9

Using Social Network Analysis to Investigate the Collaboration Between Architects and Agile Teams: A Case Study of a Large-Scale Agile Development Program in a German Consumer Electronics Company

Ömer Uludağ(✉), Martin Kleehaus, Soner Erçelik, and Florian Matthes

Technische Universität München (TUM),
85748 Garching bei München, Germany
{oemer.uludag,martin.kleehaus,soner.ercelik,matthes}@tum.de

Abstract. Over the past two decades, agile methods have transformed and brought unique changes to software development practice by strongly emphasizing team collaboration, customer involvement, and change tolerance. The success of agile methods for small, co-located teams has inspired organizations to increasingly use them on a larger scale to build complex software systems. The scaling of agile methods poses new challenges such as inter-team coordination, dependencies to other existing environments or distribution of work without a defined architecture. The latter is also the reason why large-scale agile development has been subject to criticism since it neglects detailed assistance on software architecting. Although there is a growing body of literature on large-scale agile development, literature documenting the collaboration between architects and agile teams in such development efforts is still scarce. As little research has been conducted on this issue, this paper aims to fill this gap by providing a case study of a German consumer electronics retailer's large-scale agile development program. Based on social network analysis, this study describes the collaboration between architects and agile teams in terms of architecture sharing.

Keywords: Large-scale agile development · Social network analysis · Agile architecture

1 Introduction

Emerging in the 1990s, agile methods have transformed and brought unprecedented changes to software development practice by strongly emphasizing change tolerance, continuous delivery, and customer involvement [1,2]. With these agile methods, self-organizing teams work closely with business customers in a single-project context, maximizing customer value and quality of delivered software

P. Kruchten et al. (Eds.): XP 2019, LNBIP 355, pp. 137–153, 2019.
https://doi.org/10.1007/978-3-030-19034-7_9

product through rapid iterations and frequent feedback loops [1]. The success of agile methods for small, co-located teams has inspired enterprises to increasingly apply agile practices to large-scale endeavors [2,3]. Since the initial application of agile methods was originally intended for small, co-located teams, many organizations are uncertain how to introduce them at scale and therefore face new challenges such as inter-team coordination, dependencies to other existing environments or distribution of work without a defined architecture [1,4,5]. The latter is also the reason why large-scale agile development has been subject to criticism since it neglects detailed assistance on software architecting [2,6]. Agile methods assume that architecture should evolve incrementally rather than being imposed by some direct structuring force (emergent architecture) [7]. However, the practice of this design is effective at team level but insufficient at large-scale. It causes excessive redesign efforts, architectural divergence, and functional redundancy increasing a system's complexity [7,8]. Therefore, an intentional architecture is required, which embraces architectural guidelines that specify inter-team design and implementation synchronization [7,9]. The effective evolution of a system's architecture requires the right balance of emergent and intentional architecture and a close collaboration between architects and agile teams [7,9,10].

Literature describing the collaboration between architects and agile teams in large-scale agile development is still scarce. This paper aims to fill this gap by providing a case study of a German consumer electronics retailer's large-scale agile development program. Based on this objective, our research question is:

> *How does the collaboration take place between architects and agile teams in a large-scale agile development program?*

The remainder of this paper is structured as follows. In Sect. 2, we provide an overview of foundations and related works. In Sect. 3, we present the research approach of this paper. Section 4 describes the case study on the collaboration between architects and agile teams in the large-scale agile development program. We discuss our lessons learned in Sect. 5 before concluding the paper with a summary of our results and remarks on future research in Sect. 6.

2 Background and Related Work

In the following, the Scaled Agile Framework and Spotify Model are introduced, as the observed program has adopted these two scaling frameworks. Thereafter, the concept of communication networks is presented, which is essential for interpreting the results of the social network analysis in Sect. 4.

2.1 Scaled Agile Framework

The Scaled Agile Framework (SAFe), a widely used scaling framework [11], was first published by Dean Leffingwell in 2011. SAFe builds on existing lean and agile principles that are combined into a method for large-scale agile projects.

It provides a soft introduction to the agile world as it specifies many structured patterns. This introduction is needed for organizations moving from traditional to agile development environment [7]. The latest SAFe 4.6 version supports four out-of-the-box configurations: *Essential SAFe*, *Large Solution SAFe*, *Portfolio SAFe*, and *Full SAFe*. As the observed program uses Essential SAFe, we will subsequently focus on this. Essential SAFe is the simplest entry point for implementing SAFe and consists of team and program levels [7]. At team level, the techniques outlined are those used in Scrum. Each team consists of five to nine members, one scrum master (SM), and one product owner (PO). All teams are part of an agile release train (ART), a team of agile teams that delivers a continuous flow of incremental releases. Each team is responsible for defining, building, and testing stories from its team backlog in a series of two-week iterations using common iteration cadences [7]. At program level, the product management (PM) serves as the content authority for the ART and is accountable for identifying program backlog priorities. The PM works with POs to optimize feature delivery and direct their work at team level. A release train engineer (RTE) facilitates program execution, escalates impediments, manages risk, and helps to drive continuous improvement [7]. The system architect has the technical responsibility for the overall architectural design of the system and aligns the ART with the common technical and architectural vision [7].

2.2 Spotify Model

In 2012, Kniberg and Ivarsson [12] published Spotify's approach to scale agile methods over 30 teams across three cities. The Spotify Model emphasizes the importance of "aligned autonomy", i.e. the autonomy of agile teams with simultaneous collaboration and coordination to achieve the same goals. The basic unit of development is called a *Squad*, which is similar to an agile team in SAFe. Squads are self-organizing and autonomous teams that have all the skills to design, develop, test, and release for production. A *Tribe* is designed as a collection of squads working in related areas (correspondents to an ART in SAFe). Squads within a tribe are co-located. People with similar skills in the same competency area within the same tribe form a *Chapter*. A *Guild* is a community of people that share same interests and often includes all chapters working in this area (complies with a community of practice in SAFe) [12].

2.3 Communication Networks

According to Guo and Sanchez [13], communication is understood as the creation or exchange of thoughts, ideas, and emotions between senders and receivers. Communication can be decomposed into two types: inter-team and intra-team communication. The former stands for communication between several teams, the latter for communication within a team [14]. The flow of communication connecting senders and receivers are called communication networks [15]. Figure 1 depicts five common communication networks. The *wheel network* is the most centralized network pattern. In this network, each member communicates with

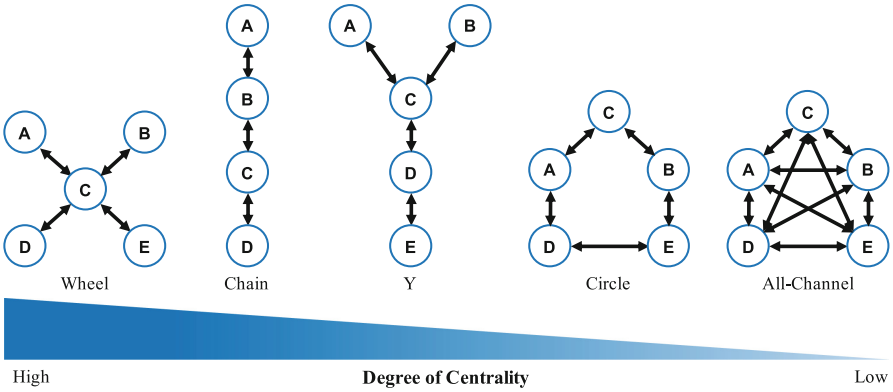


Fig. 1. Common communication networks [15]

only one other person. The superintendent C receives all the information from his subordinates A, B, D, and E and sends back information, usually in the form of decisions. The *chain network* is the second highest in centralization. Only two people communicate with each other, and they have only one other person to communicate with. The *Y network* is similar to the chain network except that two members are out of the chain. In the Y network, members A and B can send information to C but they cannot receive information from anyone else. Members C and D can exchange information. Member E can exchange information with member D. The *circle network* stands for horizontal and decentralized communication, which offers equal communication possibilities for every member. Each can communicate with one other to his right and left. Members have identical restrictions but the circle is a less restricted condition than the wheel, chain, or Y network. The *all-channel network* is an extension of the circle network and connects everyone in the circle network, as it permits each member to communicate freely with all other persons [15].

2.4 Agile Architecture

Angelov et al. [16] describe the role of architects and challenges they face in Scrum such as insufficient collaboration, lack of understanding of the value of architecture, and poor communication between team architects [16]. Bachmann et al. [17] and Nord et al. [18] present four tactics to achieve agility at scale by aligning the system architecture, organization structures and product infrastructures. These include vertical and horizontal system decomposition, matrix and augmented team structures, architecture and infrastructure runway, and deployability tactics and can be used in different phases in a system's life cycles. Uludağ et al. [10] describes how the adoption of domain-driven design supported a large-scale agile development program with three agile teams at a large insurance company. Uludağ et al. [10] report that agile teams and project managers involved in the program conceived that without any form of architectural guidance, large-scale agile development programs can hardly be successful. Dingsøyr et al. [19] investigated a large-scale development program with an extensive use

of Scrum and a focus on customer involvement, inter-team coordination, and software architecture. Two key findings related to software architecture are the tension between up-front and emergent architecture and the demanding role of architects in large-scale agile development.

3 Case Study Design

A case study is a suitable research methodology for this paper, since it helps to study contemporary phenomena in a real life context [20]. We followed the guidelines described by Runeson and Höst [20].

Case Study Design: The main objective of this paper is to investigate the collaboration between architects and agile teams in large-scale agile development in terms of architecture sharing. Based on this objective, we defined one research question (see Sect. 1). The study is a an exploratory single case study, since this paper looks into an unexplored phenomenon and aims to seek new insights and generate ideas for future research [20]. The case was purposefully selected, because the studied company has successfully adopted SAFe for building complex software for the last one and a half years. The unit of analysis is the consumer electronics retailer's large-scale agile development program.

Data Collection: We used a mixed methods approach with three levels of data collection techniques [21]. As direct methods, we observed two Program Increment (PI) Planning events [7] with low degree of interaction by the researcher and low awareness of being observed [20]. These observations provided a deep understanding of the overall structure. With the help of seven semi-structured interviews, roles and practices related to architecture were identified and documented. Quantitative data was collected by the online-survey tool Questback for building the social networks and revealing the collaboration between architects and agile teams (see Sect. 4). Therein, we asked respondents how often they exchange architectural advice and decisions with their colleagues, how often they see their colleagues, and if they have suggestions on how to improve the exchange among team members (using a Likert scale). A total of 32 out of 62 available people from eight teams took part in the survey. Three persons were removed from the analysis because no clear assignment to these persons could be made. The response rate for the remaining 29 program members from eight teams is 47% with 758 connections for architecture sharing.

Data Analysis: Interviews and observation protocols were coded using a deductive approach as proposed by Cruzes and Dybå [22]. Qualitative data collected in interviews form the theoretical foundation for interpreting social relations between architects and agile teams. After initial coding, codes were refined and consolidated by merging related ones and removing duplicates. Quantitative data was analyzed through the use of social network analysis, which comprises a set of methodological techniques that aim to describe and explore patterns in relationships that individuals and groups form with each other [23].

4 Results

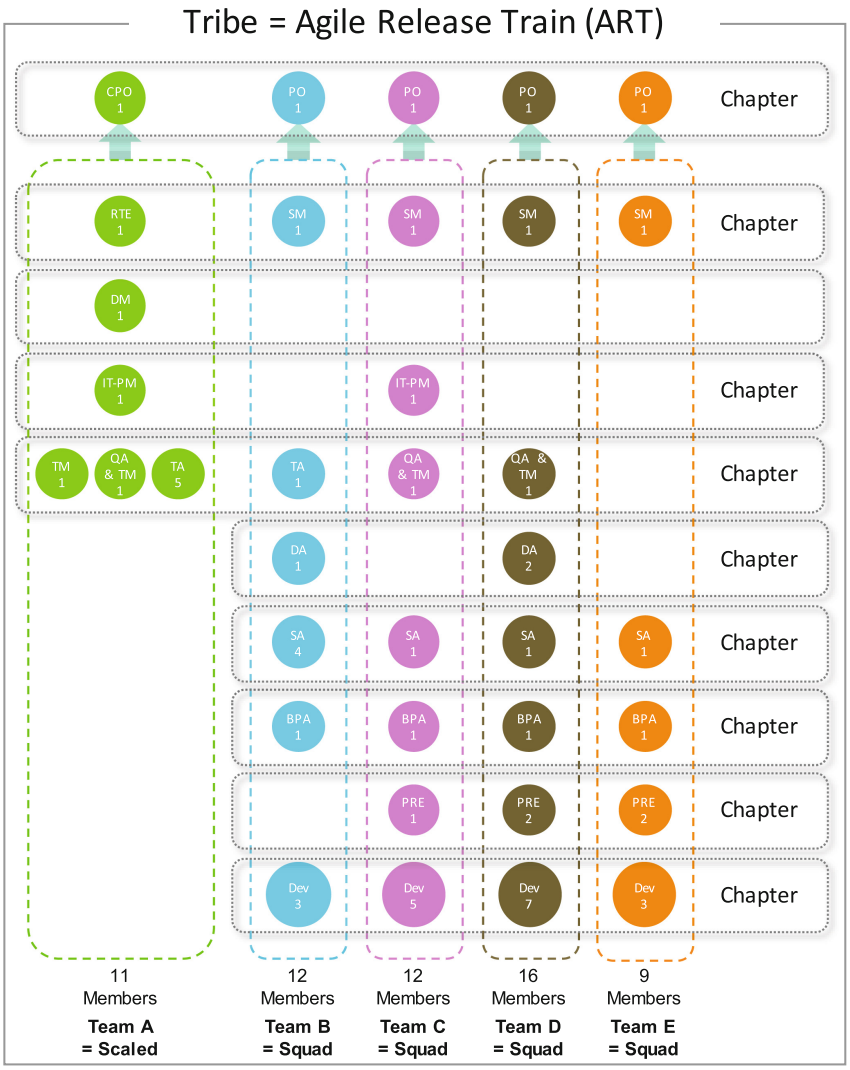
4.1 Case Description

In 2016, the case organization decided to relaunch a failed CRM project using agile methods. Due to the complexity of the project, the management decided to relaunch it with the help of a scaling framework. During early stages of research, the reasons for using Essential SAFe (from now on SAFe) became more apparent and convincing to the management. One reason for choosing SAFe was that it has proven itself in large organizations and offers comprehensive documentation. The adoption was initiated with a pilot project, which was geographically distributed. At the beginning, the pilot project faced a lot of problems. Thus, all involved employees were trained upon agile methods and SAFe by external agile coaches. After a few PIs, the responsible management team perceived that SAFe did not provide sufficient guidance on the coordination of their agile teams. Thus, the organization decided to combine SAFe with the Spotify Model. Within the transformation process, program members were divided into tribes, chapters, squads, and guilds. Figure 2 shows the current organizational structure of the observed program. Figure 2 also shows all 62 members forming a tribe. This tribe consists of a "scaled" team (Team A), which does not play a hierarchical superior, but a more coordinating role without personnel management, and four squads (Team B, Team C, Team D, and Team E). Team F, Team G, and Team H, which are not shown in Fig. 2 constitute representatives of three suppliers that provide external support for their third-party systems. The tribe is divided horizontally into nine chapters for: (1) the chief product owner (CPO) and POs, (2) RTE and SMs, (3) IT project managers (IT-PMs), (4) quality analysts and test managers (QAs & TMs), (5) data analysts (DAs), (6) solution architects (SAs)[1], (7) business process architects (BPAs), (8) product reliability engineers (PRE), and (9) developers (Devs). Each SA is assigned to a squad and takes care of the overall system architecture with its subsystems and interfaces. The team concentrates on the cross-system data flows and processes related to the integration of the architecture. These data flows and processes are used to define minimum interface requirements that all teams must meet. In contrast to SAs, who represent technical architects, BPAs are functional architects that are also dedicated to squads. The responsibilities of BPAs are not really known yet, as their role has been added to the program just recently. However, both architect roles should play a dual role within their squads by making architectural decisions and guiding them to fulfill the required architectural standards. Due to ongoing transformation, guilds have not yet been established but will be organized soon. In the following two sections, the inter- and intra-team exchange of architecture-related information of the observed program will be presented.

[1] The role of the SA in the case organization correspondents to the role of the system architect as described by SAFe [7]. For reasons of consistency, we use the same terminology as the case organization.

4.2 Inter-Team Architecture Sharing

Figure 3 provides an overview of how architecture-related information is shared across all teams. An interesting finding here is that the scaled team is located in the center of the graph. This indicates continuous communication and coordination between the scaled team and the four squads on architectural topics. Figure 3 also shows a close collaboration between Team B and Team E and between Team B and Team D, which is due to architectural dependencies between the systems on which they work. Figure 3 also provides an overview of roles that are inten-



Abbreviations	
BPA - Business Process Architect	PRE - Product Reliability Engineer
CPO - Chief Product Owner	QA & TM - Quality Analyst & Test Manager
DA - Data Analyst	RTE - Release Train Engineer
Dev - Developer	SA - Solution Architect
DM - Delivery Manager	SM - Scrum Master
IT-PM - IT Project Manager	TA - Test Analyst
PO - Product Owner	TM - Test Manager

Fig. 2. Organizational structure of the observed large-scale agile development program

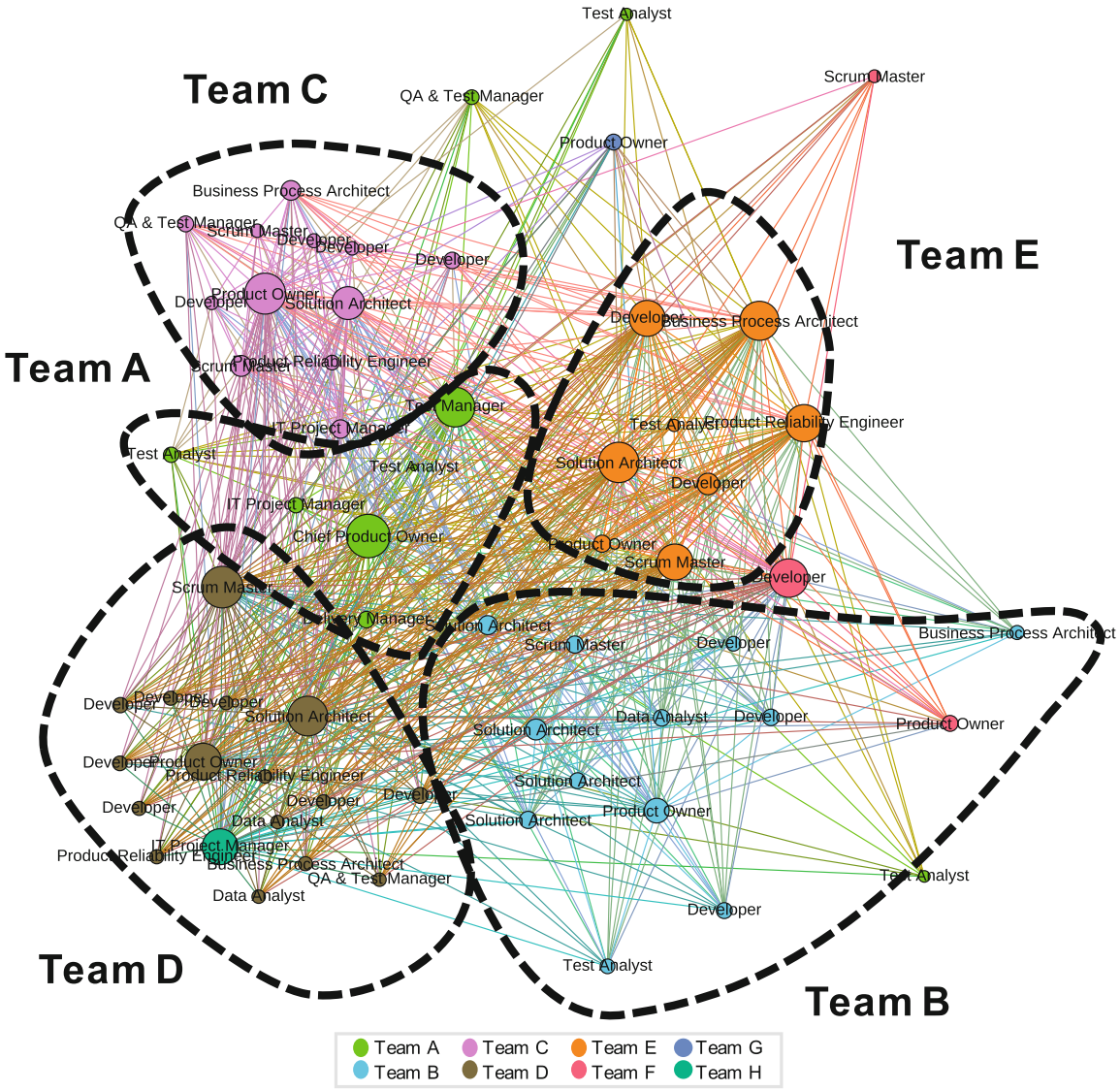


Fig. 3. Social network of eight teams including salient roles that are intensively involved in inter- and intra-team architecture-sharing

sively involved (large nodes) in architecture sharing. First, it shows that the CPO of Team A (CPO_A) is the most outstanding node in the inter- and intra-team exchange of architecture-related information. Second, SAs also form relatively large nodes compared to other roles. This observation confirms the importance of SAs for the exchange of inter- and intra-team architectural information. Figure 3 also shows that the TM_A also plays an important role in architecture sharing. Table 1 presents top 10 stakeholders involved in inter-team sharing based on the normalized degree centrality[2] measure. Table 1 shows that the CPO_A has a normalized degree centrality value of 1,0, which indicates that he/she is sharing information with all stakeholders involved in the observed program. The SA_E

[2] The normalized degree centrality is defined as the number of links of an stakeholder divided by the maximal possible number.

Table 1. Top 10 stakeholders involved in inter-team architecture sharing based on normalized degree centrality

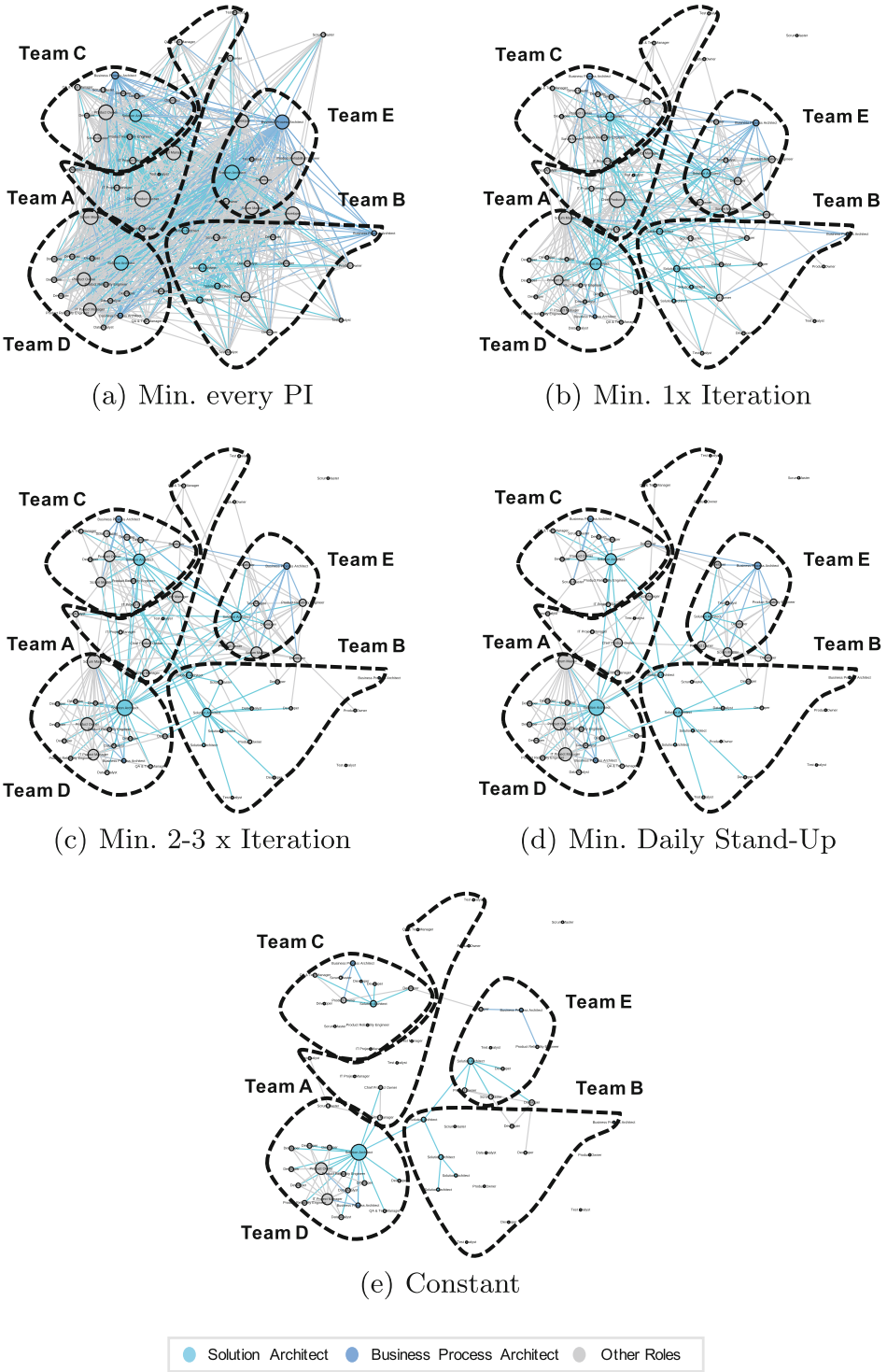
Rank	1	2	3	4	5	6	7	8	9	10
Role	CPO_A	SM_D	PO_C	SA_E	SA_D	TM_A	BPA_E	Dev_F	PRE_E	PO_D
Value	1,0	0,95	0,93	0,92	0,90	0,90	0,89	0,87	0,84	0,82

and SA_D have normalized degree centrality values of 0,92 and 0,90 indicating high involvement in inter-team sharing.
The PI planning event of SAFe is a face-to-face event [7] that aims to align all agile teams within the ART to share the common mission and vision by creating iteration plans and team objectives for the upcoming PI. It is conducted every two and a half months and offers a platform for the exchange of general and architectural information across teams, since all members of the ART are present in one location. Figure 4(a) shows that SAs and BPAs have a very strong sharing with other teams during the PI planning. Figure 4(d) reveals a chain communication between the SA_B, SA_C, SA_D, and SA_E on a daily basis. In particular, the chain is composed as follows: SA_E exchanges information with SA_B, who exchanges information with SA_D, who shares information with SA_C. This communication pattern characterizes a centralized communication between SAs. The chain communication pattern can also be observed with SA_B, SA_D, and SA_E. Figure 4(e) shows that SA_B, SA_D, and SA_E constantly[3] exchange information and that the SA_C is no longer involved in an exchange with other SAs. Figure 4 shows that SAs form a decentralized all-channel communication pattern. This means that each SA speaks with all other SAs. The overall comparison also shows that the three external SA of Team B are less participating in the inter-team exchange than the rest of internal SAs involved in the program. Other roles such as SM, TM, PO, and CPO are also heavily involved in exchange of information within the PI planning. The shorter the observed time intervals become, the more dominant the SA becomes with regards to the inter-team sharing.

4.3 Intra-Team Architecture Sharing

The exchange of architectural information in Team B shows a central wheel communication pattern between SAs, since external SAs are guided by the internal SA, who represents the intra-team lead architect (see Fig. 5(a)). Figure 5(a) also shows that SAs form the core of the team. Moreover, Fig. 5(a) shows that BPA_B only exchanges information with another role. A decentralized all-channel communication pattern can be observed in Team C (see Fig. 5(b)). This means that other non-architectural roles exchange information without necessarily involving SA_C. Nevertheless, SA_C plays the most central role, since the SA frequently communicates with all team members. Compared to BPA_B, BPA_C plays a more central role, as he/she shows a close collaboration and communication with his/her

[3] Constant exchange means that it takes place more than once a day.



(a) Min. every PI (b) Min. 1x Iteration

(c) Min. 2-3 x Iteration (d) Min. Daily Stand-Up

(e) Constant

Fig. 4. Social networks focusing on SAs and BPAs with regards to the frequency of inter- and intra-team architecture sharing

squad (see Fig. 5(a) and (b)). The comparison of the two figures also shows that SA_C and BPA_C exchange information more frequently than SA_B and BPA_B. Figure 5(b) shows a decentralized all-channel communication pattern between architects and other team members of Team D. Similar to BPA_C, BPA_D often

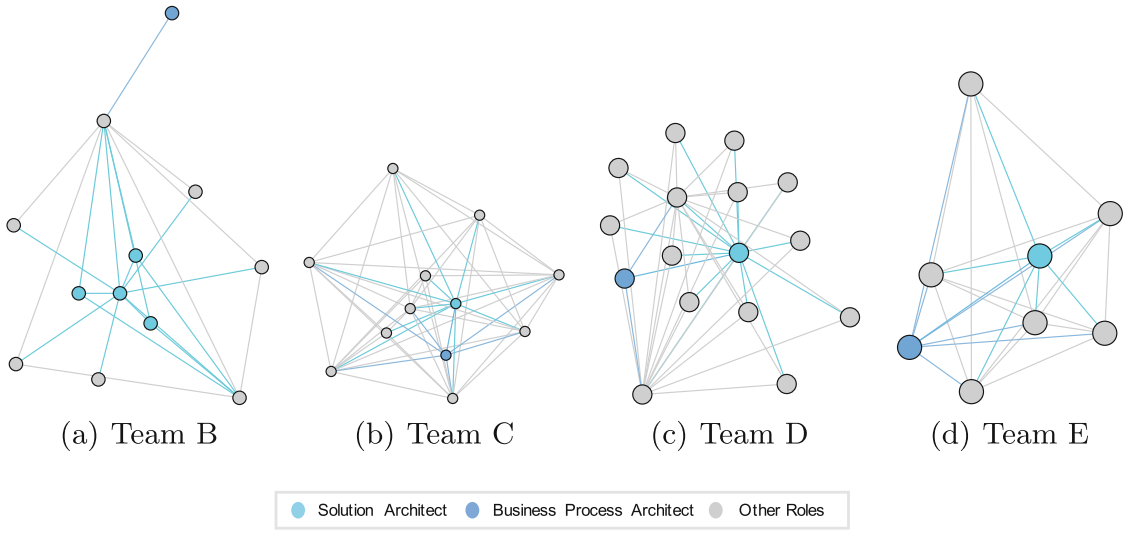


Fig. 5. Social network of the four squads focusing on SAs and BPAs involved in intra-team architecture-sharing

Table 2. Normalized degree centralities of architects in intra-team architecture sharing

	Team B	Team C	Team D	Team E
SA	0,91	1,0	1,0	1,0
BPA	0,09	0,64	0,2	1,0

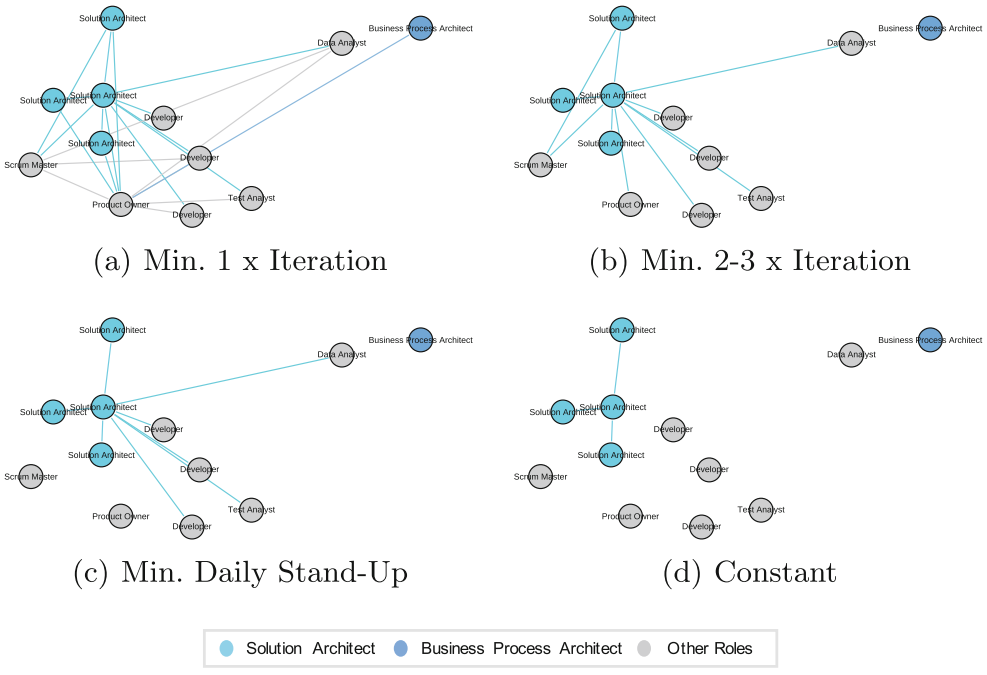
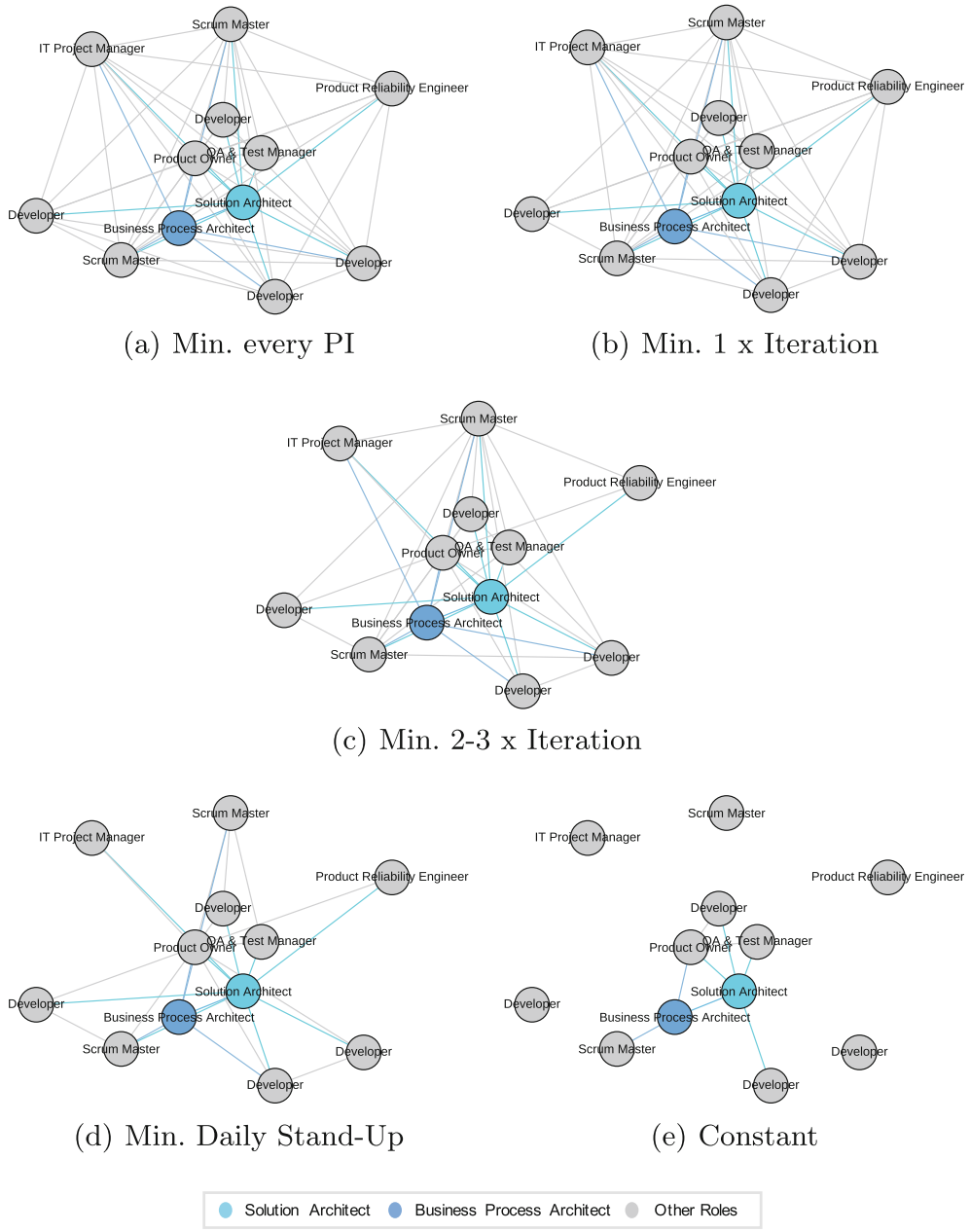


Fig. 6. Social network of Team B focusing on SAs and BPAs with regards to the frequency of intra-team architecture sharing



(a) Min. every PI (b) Min. 1 x Iteration

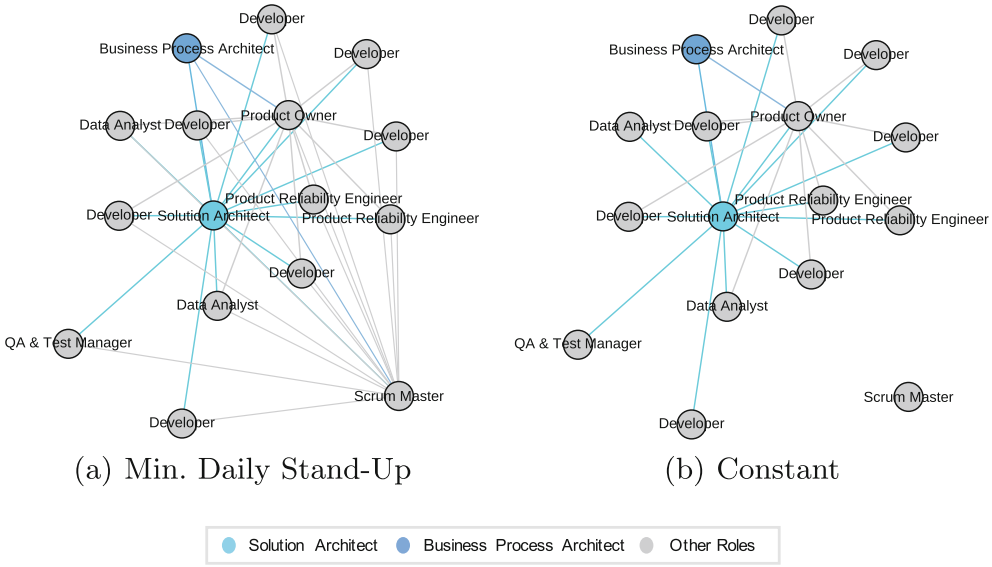
(c) Min. 2-3 x Iteration

(d) Min. Daily Stand-Up (e) Constant

Fig. 7. Social network of Team C focusing on SAs and BPAs with regards to the frequency of intra-team architecture sharing

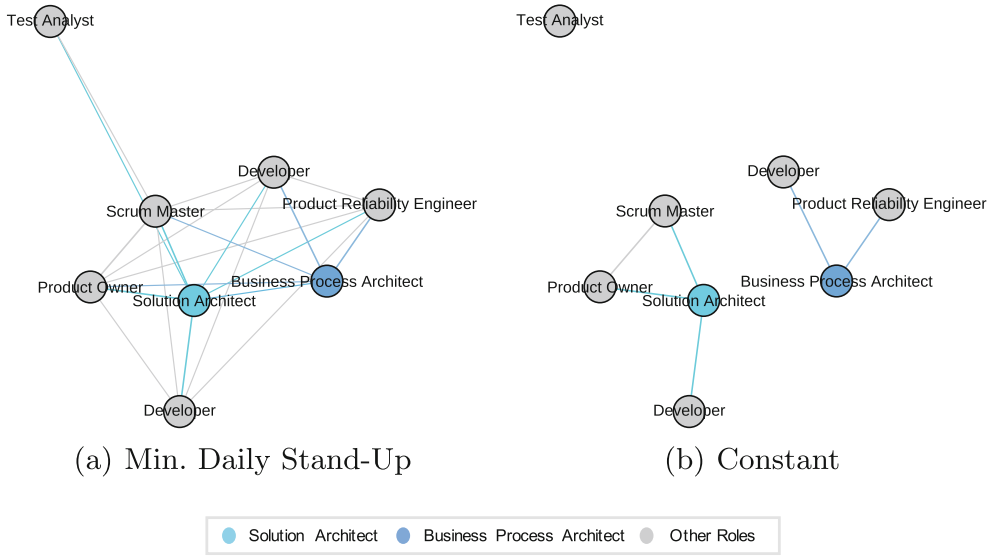
exchanges architecture information with team members. Table 2 shows the normalized degree centrality values of SAs and BPAs involved in intra-team architecture sharing. 75% of the SAs possess a normalized degree centrality value of 1,0 indicating that they share information with all squad members. Comparing SAs with BPAs, Table 2 shows that SAs have a stronger exchange of information with their squad members than BPAs (except Team E).

Figure 6 shows how Team B's intra-team sharing changes at four distinct time intervals. For instance, Fig. 6(a) shows that BPA_B only exchanges information with one Dev_B once per iteration. Figure 6(b) shows that the exchange of information between SAs and non-architectural roles mostly takes place two to three



(a) Min. Daily Stand-Up (b) Constant

Fig. 8. Social network of Team D focusing on SAs and BPAs with regards to the frequency of intra-team architecture sharing



(a) Min. Daily Stand-Up (b) Constant

Fig. 9. Social network of Team E focusing on SAs and BPAs with regards to the frequency of intra-team architecture sharing

times per iteration, while the sharing between SAs takes place constantly (see Fig. 6(d)). Similar to Figs. 6, 7 shows Team C's intra-team architecture sharing. The exchange in the team usually takes place two to three times per iteration (see Fig. 7(c)). Sharing between architects and non-architectural roles takes place on a daily basis (see Fig. 7(d)). In contrast to Team B, Fig. 7(e) shows that SA_C and BPA_C constantly communicate together. Figure 8(a) shows that the exchange between architects and non-architectural roles as well as among architects mainly takes place on a daily basis. SA_D and BPA_D constantly exchange architectural information (see Fig. 8(b)). Figure 8(b) also shows that other mem-

bers such as DA_D, QA & TM_D, PRE_D, and PO_D constantly exchange architectural information. The intra-team exchange of Team E takes place mainly on a daily basis (see Fig. 9(a)). SA_E and BPA_E communicate on a daily basis (see Fig. 9(b)). Architecture sharing between architects and non-architectural roles takes place on a daily basis. Figure 9(b) shows that two groups are formed during the constant exchange of information. The first group includes SA_E, SM_E, Dev_E, and PO_E, while the second group constitutes Dev_E, BPA_E, and PRE_E. Table 3 provides a summary of the social network analysis with identified communication patterns and frequencies.

5 Discussion

5.1 Key Findings

Both architectural roles, i.e. SAs and BPAs, and other roles, e.g. TMs, SMs, and POs, are involved in inter- and intra-team architecture sharing. In particular, the CPO plays one of the most salient roles. An all-channel communication network can be observed in each squad. SAs enable a decentralized exchange so that other team members can exchange architecture-relevant information without necessarily involving SAs. This observation coincides with the values and principles of agile software development. Both SAs and BPAs prefer face-to-face communication with their team members and do not exchange information by including bridging roles. Each squad is accompanied by at least one SA and BPA. Both architects play a dual role in their squads. On the one hand, they make architectural decisions and iteratively create architecture models. On the other hand, they provide guidance and support their squad in meeting architectural standards. With this setup, the observed program aims to increase development speed by balancing emergent and intentional architecture. In all social networks, SAs form central nodes in inter- and intra-team sharing.

Table 3. Summary of the social network analysis

	Team A	Team B	Team C	Team D	Team E
Overview (communication network)	decentralized; -	decentralized; all-channel	decentralized; all-channel	decentralized; all-channel	decentralized; all-channel
Architects with other roles (communication network)	not applicable	decentralized; all-channel	decentralized; all-channel	decentralized; all-channel	decentralized; all-channel
Architects with other roles (frequency of sharing)	not applicable	mostly 2-3x pro iteration (except BPA)	mostly daily basis	mostly daily basis	mostly daily basis
Between architects (Communication Network)	not applicable	centralized; wheel (between SAs)	not applicable	not applicable	not applicable
Between architects (frequency of sharing)	not applicable	constant (except BPA)	constant	constant	mostly daily basis

5.2 Threats to Validity

We discuss potential threats to validity along with an assessment scheme as recommended by Runeson and Höst [20].

Construct Validity: This aspect reflects to what extent operational measures that are studied really represent what the researcher has in mind [20]. Two countermeasures were taken for construct validity. First, interview protocols were coded by the author of this paper and reviewed by a second researcher. Second, a key informant of the organization has reviewed the analyses of this paper.

Internal Validity is irrelevant, as this study was neither explanatory nor causal.

External Validity: This aspect of validity concerns to what extent the findings can be generalized, and to what extent the findings are of interest to other persons outside the case under investigation [20]. This paper focuses on analytical generalization [20] by providing a detailed description of the case. It provides empirical insights that allow for a profound understanding on the collaboration between architects and agile teams. The shown findings should be viewed as valuable insights for other organizations that adopted Essential SAFe.

Reliability: This validity is concerned with to what extent the data and the analysis are dependent on the specific researcher [20]. To mitigate this threat, two countermeasures were taken. First, the case study has been designed so that the large number of interviewees and multiple interviewers allowed data and observer triangulation. Second, a case study database was created, which includes case study documents such as audio recordings protocols, and field notes of observations.

6 Conclusion and Future Work

In this paper, we described the collaboration between architects and agile teams in a large-scale agile development program of a German consumer electronics retailer. Due to the complexity and extent of the CRM product, each squad is guided and supported by at least one SA and BPA. Each SA is responsible for the architecture of a subsystem and ensures that the respective squad complies with defined architectural requirements. The observed program also introduced the new role of the BPA that is responsible for developing the functional architecture of the subsystem. To understand the role of SAs and BPAs and their collaboration with squads, we investigated social networks of one scaled team and four squads. We learned that intra-team architecture sharing is usually facilitated by SAs. Comparing the social networks with common communication networks, we discovered that SAs and BPAs prefer direct communication. For the most part, architects share information on a daily basis with their teams. The intra-team sharing between architects and their teams is characterized by an all-channel communication network.

As future work, we will continue to study the large-scale agile development program of the German consumer electronics retailer. First, we will research how

the current state of architecture sharing is perceived by the stakeholders and how it could be improved by the use of various coordination mechanisms such as ad hoc meetings, co-location or communities of practices. Second, as the squads in the large-scale agile development program become more mature and evolve towards feature teams, we will investigate the architectural decision-making process of squads. We hope to gain a better understanding of the collaboration between architects and squads regarding the distribution of their responsibilities for architectural issues.

References

1. Kettunen, P.: Extending software project agility with new product development enterprise agility. Softw. Process: Improv. Pract. **12**(6), 541–548 (2007)
2. Dingsøyr, T., Moe, N.B.: Towards principles of large-scale agile development. In: Dingsøyr, T., Moe, N.B., Tonelli, R., Counsell, S., Gencel, C., Petersen, K. (eds.) XP 2014. LNBIP, vol. 199, pp. 1–8. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-14358-3_1
3. Alqudah, M., Razali, R.: A review of scaling agile methods in large software development. Int. J. Adv. Sci. Eng. Inf. Technol. **6**(6), 28–35 (2016)
4. Dikert, K., Paasivaara, M., Lassenius, C.: Challenges and success factors for large-scale agile transformations: a systematic literature review. J. Syst. Softw. **119**, 87–108 (2016)
5. Uludag, Ö., Kleehaus, M., Caprano, C., Matthes, F.: Identifying and structuring challenges in large-scale agile development based on a structured literature review. In: IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC) 2018, pp. 191–197. IEEE (2018)
6. Rost, D., Weitzel, B., Naab, M., Lenhart, T., Schmitt, H.: Distilling best practices for agile development from architecture methodology. In: Weyns, D., Mirandola, R., Crnkovic, I. (eds.) ECSA 2015. LNCS, vol. 9278, pp. 259–267. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23727-5_21
7. Leffingwell, D.: SAFe® 4.5 Reference Guide: Scaled Agile Framework® for Lean Software and Systems Engineering. Addison-Wesley Professional, Boston (2018)
8. Mocker, M.: What is complex about 273 applications? untangling application architecture complexity in a case of european investment banking. In: 2009 42nd Hawaii International Conference on System Sciences 2009, HICSS, pp. 1–14. IEEE (2009)
9. Waterman, M.: Reconciling agility and architecture: a theory of agile architecture, Ph.D. thesis, Victoria University of Wellington (2014)
10. Uludağ, Ö., Hauder, M., Kleehaus, M., Schimpfle, C., Matthes, F.: Supporting large-scale agile development with domain-driven design. In: Garbajosa, J., Wang, X., Aguiar, A. (eds.) XP 2018. LNBIP, vol. 314, pp. 232–247. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91602-6_16
11. Uludağ, Ö., Kleehaus, M., Xu, X., Matthes, F.: Investigating the role of architects in scaling agile frameworks. In: 2017 IEEE 21st International Enterprise Distributed Object Computing Conference (EDOC), pp. 123–132. IEEE (2017)
12. Kniberg, H., Ivarsson, A.: Scaling agile @ spotify (2012)
13. Guo, L.C., Sanchez, Y.: Workplace communication. Organizational behavior in health care, pp. 77–110 (2005)

14. Presbitero, A., Roxas, B., Chadee, D.: Effects of intra- and inter-team dynamics on organisational learning: role of knowledge-sharing capability. Knowl. Manag. Res. Pract. **15**(1), 146–154 (2017)
15. Lunenburg, F.: Network patterns and analysis: underused sources to improve communication effectiveness. Nat. Forum Educ. Adm. Super. J. **28**(4), 1–7 (2011)
16. Angelov, S., Meesters, M., Galster, M.: Architects in scrum: what challenges do they face? In: Tekinerdogan, B., Zdun, U., Babar, A. (eds.) ECSA 2016. LNCS, vol. 9839, pp. 229–237. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48992-6_17
17. Bachmann, F., Nord, R.L., Ozakaya, I.: Architectural tactics to support rapid and agile stability. Technical report, Carnegie-Mellon University Pittsburgh PA Software Engineering Institute (2012)
18. Nord, R.L., Ozkaya, I., Kruchten, P.: Agile in distress: architecture to the rescue. In: Dingsøyr, T., Moe, N.B., Tonelli, R., Counsell, S., Gencel, C., Petersen, K. (eds.) XP 2014. LNBIP, vol. 199, pp. 43–57. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-14358-3_5
19. Dingsøyr, T., Moe, N.B., Fægri, T.E., Seim, E.A.: Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation. Empirical Softw. Eng. **23**(1), 490–520 (2018)
20. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical softw. Eng. **14**(2), 131 (2009)
21. Lethbridge, T.C., Sim, S.E., Singer, J.: Studying software engineers: data collection techniques for software field studies. Empirical softw. Eng. **10**(3), 311–341 (2005)
22. Cruzes, D.S., Dyba, T.: Recommended steps for thematic synthesis in software engineering. In: 2011 International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 275–284. IEEE (2011)
23. Scott, J.: Social Network Analysis. Sage, Thousand Oaks (2017)

What to Expect from Enterprise Architects in Large-Scale Agile Development? A Multiple-Case Study

Completed Research

Ömer Uludağ
Technische Universität München
oemer.uludag@tum.de

Martin Kleehaus
Technische Universität München
martin.kleehaus@tum.de

Niklas Reiter
Technische Universität München
niklas.reiter@tum.de

Florian Matthes
Technische Universität München
matthes@tum.de

Abstract

In modern times, traditional enterprises are confronted with rapidly changing customer demands, increasing market dynamics, and continuous emergence of technological advancements. Confronted with the imperatives of a digital world, companies are striving to adopt agile methods on a larger scale to meet these requirements. In recent years, enterprise architecture management has established itself as a valuable governance mechanism for coordinating large-scale agile transformations by connecting strategic considerations to the execution of transformation projects. Our research is motivated by the lack of empirical studies on the collaboration between enterprise architects and agile teams. Against this backdrop, we present a multiple-case study of five leading German companies that aims to shed light on this field of tension. Based on our results from 20 semi-structured interviews, we present the expectations of agile teams for enterprise architects and how they are fulfilled.

Keywords

Enterprise architecture management, large-scale agile development, multiple-case study

Introduction

In increasingly hypercompetitive environments, enterprises face a multitude of challenges such as continuously changing customer demands, rapidly evolving technological advancements, and tensing regulatory uncertainties (Fuchs and Hess 2018; Kettunen and Laanti 2017). Consequently, companies are urged to undergo organizational transformations to respond readily to environmental changes (Besson and Rowe 2012; Gerster et al. 2018). To master their respective organizational transformation, firms are extensively adopting agile practices which often necessitate large-scale agile transformations (Dikert et al. 2016; Fuchs and Hess 2018). However, these transformations entail new managerial challenges (Dingsøyr et al. 2018) such as hierarchical organizational structures that prevent the wide adoption of agile (Hekkala et al. 2017), traditional project management mechanisms that are overly process driven and bureaucratic (Gregory et al. 2015), and coordination and alignment issues between large-scale agile activities as well as between agile and non-agile teams (Scheerer et al. 2014). The term "large-scale agile development" has been used to describe multi-team development efforts that make use of agile principles involving a high number of actors and interfaces with existing systems (Dingsøyr et al. 2014; Rolland et al. 2016).

In recent years, enterprise architecture management (EAM) has established itself as a valuable governance mechanism to coordinate transformations by connecting strategic considerations to the execution of large-scale agile projects (Greefhorst and Proper 2011). It covers all dimensions of an enterprise (business, application, information, and infrastructure aspects) and fosters the mutual alignment of business and IT (Hauder et al. 2014; Rouhani et al. 2015).

In large-scale agile development, typical tasks of enterprise architects (EAs) include (1) harmonizing governance requirements across sprints and agile teams (ATs), (2) supporting ATs by aligning individual project strategies with enterprise objectives, and (3) working closely with ATs by guiding them through business and technical roadmaps (Uludağ et al. 2017). However, the mutual expectations of EAs and ATs are not always frictionless (Kulak and Li 2017). This tension originates to some extent from the antithetic ways of thinking and mindsets of both stakeholder groups. EAs embrace a top-down perspective focusing on long-term goals and strategies (Hanschke et al. 2015). This top-down decision model may conflict with ATs' short-term ambitions to satisfy the business representatives (Dingsøyr et al. 2018). The pressure to deliver business functionality may lead to the negligence of long-term architectural improvements (Dingsøyr et al. 2018). The traditional "*command and control*" culture of EAs may oppose the conversant "*servant leadership*" culture of ATs (Kulak and Li 2017).

Our findings are consonant with those of Canat et al. (2018), Hanschke et al. (2015), and Hauder et al. (2014) that empirical studies on the collaboration between EAs and ATs are lacking. Against this backdrop, we aim to fill this gap formulating the following research questions:

RQ 1: What expectations do agile teams have for enterprise architects?

RQ 2: To what extent are these expectations satisfied by enterprise architects?

Theoretical Background and Related Work

Agile methods such as Scrum or XP rely mainly on emergent design practices, meaning that architecture emerges from the system and is not imposed by some direct structuring force (Babar 2009). The practice of emergent design is effective at the team level but insufficient when agile methods are applied on a larger scale. For large-scale agile development, a certain degree of architectural planning and governance becomes more important (Leffingwell et al. 2008) as they ensure the alignment of ATs to achieve desirable organization-wide effects (Ovaska et al. 2003) and provide a common target vision by combing strategic considerations with the execution of agile projects (Greefhorst and Proper 2011). Alike (Greefhorst and Proper 2011), we take the view that the mutual alignment of large-scale agile transformation can be achieved by an effective EAM function. Until now, EAM initiatives mostly focused on top-down governance (Winter 2016). However, this enforcement-centric view does not fit well into increasingly widespread agile environments as it restricts the design freedom of ATs and encounters their resistance (Kulak and Li 2017). In recent years, a number of novel EAM approaches have been proposed to address the aforementioned problems, which are presented in the following.

Agile Enterprise Architecture Management (Hauder et al. 2014)

Hauder et al. (2014) propose an organization-specific agile EAM function that consists of the three main phases. In the first phase, the EAM function starts by motivating an EAM endeavor. In this phase, the EAM function must convince the stakeholders of the meaningfulness of the EAM function and its long-term benefits for the entire company. Here, the EAM function must ensure that the top management and other key stakeholders support the EAM endeavor. Next to motivating the EAM endeavor, the EAM team collects information from various stakeholders that serves as a decision base later on. The collected information is formalized by developing stakeholder-specific models. In the second phase, developed models and concepts are communicated and used to explain decisions. This phase is characterized by strong communication and supporting activities between the EAM team and its stakeholders. In this phase, the EAM team should show the turnover for each individual stakeholder. In this step, the EAM team gathers feedback by asking its stakeholders what went well and what went wrong. In the third phase, the EAM team not only analyzes and reflects on its results and practices, but also analyzes and reflects on the feedback and engagement of stakeholders. Based on a new information base, the EAM team may consider adjusting the EAM function.

Adaptive Enterprise Architecture (Wilkinson 2006)

Based on a case study, Wilkinson (2006) proposes a method for designing an adaptive Enterprise Architecture by exploiting the use of architecture principles, IT governance, and the adaptability of a service-oriented computing infrastructure. In addition, Wilkinson (2006) describes a set of architecture principles that include modularity, simplifcation, integration, and standardization that can be applied to build an adaptive Enterprise Architecture.

Collaborative Enterprise Architecture (Bente et al. 2012)

Bente et al. (2012) propose six building blocks for a collaborative Enterprise Achitecture based on lean and agile practices to tackle identified EAM problems such as dealing with the speed of change in the landscape of business models and IT systems or enforcing rules and compliance where each project has a high degree of freedom. Benefiting from examined sources paired with their professional experience, Bente et al. (2012) describe how architecture processes can be streamlined, how an agile and lean EAM initiave can be built, and how collaboration and participation can be fostered.

Lightweight Enterprise Architecture (Theuerkorn 2004)

As enterprise system landscapes are becoming increasingly complex, Theuerkorn (2004) proposes a lightweight Enterprise Architecture framework that uses a set of architectural artifacts to align IT with business strategy and to govern and evolve complex IT systems in organizations. The framework picks up the issues of many Enterprise Architecture functionalities that fail to deliver their promised benefits in time or cost. Besides an extensive explanation of the framework itself, Theuerkorn (2004) also explains how to deal with bad behaviors of architects, e.g., EAs who have their own agenda in mind and do not care about others or even perceive them as competition, thus causing trouble.

Case Study Design

Since our research is motivated by a practical problem, we apply case study research as it provides an in-depth overview of real-life situations and contemporary phenomena (Easterbrook et al. 2008). In the following, we outline the design of this multiple-case study in line with Runeson and Höst (2008).

Case Study Design: Our goal is to explore the expectations of ATs for EAs in large-scale agile development. We also seek to describe and explain the situation whenever possible. Therefore, our case study is of exploratory nature but also bears a descriptive or explanatory character (Runeson and Höst 2008). Additionally, this paper employs a multiple-case study design with five organizations that allows cross-case analysis (Yin 1994). The cases were purposefully selected because the companies undergo major agile transformations and their traditional EAM functions face unprecedented challenges while collaborating with large-scale agile development endeavors. We selected cases from diverse industries to avoid industry bias. An overview of the case organizations and conducted interviews is presented in Table 1.

Industry and code name of case organization	Head-quarter location	Company size [employees]	No. of inter-views	Position of interviewees
Global insurance company ("GlobalInsureCo")	Germany	140,000+	4	Agile developer (AD); enterprise architect (EA); chapter lead agile coaching (CLAC)
Car manufacturer ("CarCo")	Germany	130,000+	5	Chief technology officer (CTO); enterprise architect (EA); product owner (PO)
Information technology company ("ITCo")	Germany	7,000+	2	Enterprise architect (EA); product owner (PO)
Retail company ("RetailCo")	Germany	50,000+	5	Chapter lead business process architecture (CLBPA); enterprise architect (EA); product owner (PO); scrum master (SM)
Public sector insurance company ("PublicInsureCo")	Germany	6,700+	4	Agile developer (AD); enterprise architect (EA); head of IT governance department (IT-G)

Table 1. Overview and specifics of case organizations and conducted interviews

Data Collection: We focused on first- and third-degree data collection techniques (Lethbridge et al. 2005). First-degree techniques are direct methods where the researcher is in direct contact with the subject and collects data in real time. For that, we conducted 20 individual and group interviews which were semi-structured. In almost all companies, at least one senior executive, one EA, and one member of an AT were

interviewed to gain a diverse perspective on the expectations for EAs and further triangulate our results. The interviews followed a semi-structured questionnaire and were rather conversational to allow interviewees to explore their views in detail (Yin 1994). Each interview lasted 45-60 minutes and was primarily conducted in face-to-face meetings. At least two researchers were present in the interviews to facilitate observer triangulation and to mitigate the risk of researcher bias (Runeson and Höst 2008). We supplemented our interview findings by third-degree data collection techniques, which allow the researcher to get an independent analysis of work artifacts based on already available and sometimes compiled data. Slide decks and wiki pages of the cases provided us in-depth information about their EAM initiatives. Hence, we were able to obtain supplementary information on how EAs perceive their role in large-scale agile development or how EAs themselves work agile and lean. The purposeful selection of different data sources and roles from different case organizations enables the triangulation of data sources (Stake 1995).

Data Analysis: The interviews were recorded, transcribed, then coded by one researcher with open coding (Miles et al. 2014) and using the qualitative data analysis software MAXQDA[1]. Preliminary codes were consolidated and checked for consistency and completeness by another researcher. Subsequently, groups of code phrases were merged into concepts that were later related to the formulated research questions.

Results

Expectations of Agile Teams for Enterprise Architects

We used the organization-specific agile EAM practice model (Hauder et al. 2014) to determine the expectations of ATs for EAs. Our results are presented along with the agile EAM model. Here, we focus on the activities, which are particularly relevant for the collaboration between EAs and ATs.

(1) Models: In all cases, application landscape diagrams and business capability maps (sometimes referred to as domain maps) were the primary architecture models provided by EAs for ATs. Further, in all case organizations, business capability maps were enriched by additional information, e.g., applications, technologies, infrastructure components, etc. Other commonly models named were data models, interface models, system communication models or process models. Furthermore, technical reference architectures, architecture blueprints, architecture patterns or technical reference architectures were mentioned. In all organizations, interviewees stated that ATs have the following expectations of provided models: availability, binding force, quality, and relevance. Further important expectations of models mentioned were added value, applicability, and level of detail.

(2) Availability: Across all cases, the general statement was that EAs should not be a part of ATs as they should remain within their overarching role and coordinate multiple ATs. However, ATs' expect on demand availability of EAs who should support and consult ATs on architectural issues as needed. The majority of interviewees rated the availability of EAs to support as ATs as moderate, mainly due to capacity bottlenecks.

(3) Communication: The communication between ATs and EAs is mainly indirect via third roles. At CarCo, EAs and solution architects (SAs) regularly discuss and decide on cross-team architecture topics within architecture boards. The decisions made are communicated to the respective teams by SAs. However, ATs can use common communication media such as wikis, e-mails or phone to communicate with EAs. Face-to-face communication between EAs and SAs takes place frequently. At ITCo, the communication between ATs and EAs is more complicated. There, EAs align with domain architects (DAs) through so-called business area architecture meetings on a monthly basis. The topics discussed are communicated by DAs to SAs, which forward related information to their corresponding teams. Nevertheless, ATs can directly communicate with EAs via phone or e-mails. A similar flow of communication can be observed at RetailCo. EAs meet weekly with DAs and SAs in communities of practices for architecture (CoPA) and discuss overarching architecture topics. Participation is mandatory for all architects. The topics discussed there are then communicated to the teams. Again, ATs can directly communicate with EAs by scheduling face-to-face meetings. In some cases, EAs and ATs sit in the same building. The interviewees considered the co-location of EAs and ATs helpful as it increases productivity due to the shorter physical and communication distances. At PublicInsureCo, communication between EAs and ATs is in some cases direct, e.g., when EAs join ATs at the beginning of a project. The communication between the two is facilitated by daily stand-ups,

[1] https://www.maxqda.com/

plannings or retrospectives. In other cases, communication between EAs and ATs takes place indirectly via DAs and SMs. Apart from a CoPA every two weeks, there are no other regular meetings between EAs and ATs. Alike PublicInsureCo, EAs at GlobalInsureCo can join ATs as needed and directly communicate with them. However, the regular exchange is indirect through CoPAs and SAs. Like CarCo, ATs can use diverse communication tools such as e-mail or phone to contact EAs. In all cases, ATs expect a more frequent and personal contact to EAs. They also consider the communication through third persons as suboptimal.

(4) Involvement: The involvement of ATs in relevant architecture processes differs across the case organizations. At CarCo, the to-be architecture of a project is defined without involving ATs. At the beginning of a project, the to-be architecture is handed over to a SA who is responsible for intra-team architecture decisions. Deviations between the as-is and to-be architecture have to be documented and justified by the SAs. Hence, ATs are not involved in the creation of intentional architecture but have some degree of freedom for emergent architecture. At ITCo, the involvement of ATs in relevant architecture processes is accomplished by DAs who act as representatives of ATs (similar to SAs at CarCo). At RetailCo, ATs and their respective SAs take responsibility for intra-team architecture decisions. However, on higher levels, ATs are not involved in architecture processes. Similar to CarCo, ATs at PublicInsureCo and GlobalInsureCo are less involved in up-front planning but are involved in the creation of emergent architecture. Across all cases, however, ATs want to be involved in the creation of intentional architecture. One EA of PublicInsureCo argued that "*[ATs] should be very strongly involved, as they are the domain experts*". An PO of CarCo added that "*Each project should retain freedom and be accepted by EAs*".

(5) Support: Currently, EAs support ATs by providing architecture principles, assisting teams in their realization (CarCo; PublicInsureCo; GlobalInsureCo), providing tools and technology stacks (CarCo; PublicInsureCo), and guiding them through technical roadmaps (CarCo; PublicInsureCo). In addition, EAs consult and support ATs on architectural issues (RetailCo; PublicInsureCo; GlobalInsureCo) and mediate further relevant contact persons (GlobalInsureCo). In some cases, EAs perform architecture spikes (PublicInsureCo) and directly collaborate with ATs on a code basis (GlobalInsureCo). However, ATs expect first and foremost hands-on solutions and assistance in selecting new tools (CarCo; RetailCo; PublicInsureCo). Also, ATs expect personal support and guidance in the realization of the to-be enterprise architecture (ITCo; PublicInsureCo; GlobalInsureCo).

(6) Feedback: We observed that no formal and standardized feedback processes between ATs and EAs exist. In general, ATs are able to provide feedback informally and as needed. For this reason, ATs can use typical communication media such as e-mail or phone or arrange face-to-face meetings. In almost all companies, CoPAs were mentioned as an adequate arena for providing feedback (CarCo; RetailCo; PublicInsureCo; GlobalInsureCo). At CarCo and PublicInsureCo, ATs provide feedback by commenting on the wikis of EAs. On the one hand, some ATs stated that their expectations of the current form and frequency of giving EAs feedback are met: "*As it is now*" (PO, RetailCo; AD, PublicInsureCo). On the other hand, others expect of having regular appointments with EAs: "*One enterprise architecture block every week to visit 15 teams*" (SM; RetailCo). Also, ATs would like to continue giving EAs feedback through CoPAs (CarCo; RetailCo; PublicInsureCo; GlobalInsureCo) or personal dialogues (ITCo; CarCo). Across all case organizations, interviewees mentioned that feedback from ATs is reflected by EAs. Based on the feedback, EAs revise architecture principles and guidelines, adapt the meta-model of their EAM repositories to create new architecture models, provide new or revised architecture artifacts or attempt to work more closely with ATs. All respondents appreciated the EA' reflections on feedback.

Self-Perception versus External Perception of Enterprise Architects

After revealing the ATs' expectations, we asked all interviewees to rate and justify the extent to which these expectations are met by EAs.

(1) Models: All stakeholder groups felt that the architecture models provided by EAs only partially met the expectations. In general, ATs considered the models too abstract or unspecific and thus irrelevant (PO, CarCo; AT, RetailCo). One AT member enjoyed the freedom due to their high degree of abstraction: "*I enjoy the space, so the expectations are fulfilled*" (AD, PublicInsureCo). Two team members indicated that the required architecture models were not made available in time (PO, CarCo; AT, RetailCo). Interestingly, EAs had the most negative perception of the models. In this respect, four main pain points were mentioned. First, current architecture models offer a high level of technical detail and thus are considered immature (EA ITCo; EA, RetailCo; EA, PublicInsureCo). Second, EAs recognized difficulties of ATs in understanding

their architecture models because they are too abstract, general or complex (EA, RetailCo; EA, GlobalInsureCo). Third, the architecture models do not provide sufficient guidance on how they should be implemented by ATs, thus being considered not practicable (EA, RetailCo; EA, GlobalInsureCo). Last but not least, the communication of the models is considered as insufficient, leading to a low level of awareness by ATs (EA, CarCo; EA, RetailCo; EA, GlobalInsureCo). The EA of GlobalInsureCo suggested communicating architecture models when starting a new project. Managers shared the opinion that most of the models are mainly driven by standardization and complexity reduction aspects. The CTO of CarCo stated that "*one size fits all architecture models"* do not meet the expectations of ATs and that the models mostly propose solutions that include unnecessary or missing functionalities, causing additional efforts and costs. The IT-G of PublicInsureCo stated that many EAs assume that old proven models would be still useful for new problems. He suggested that EAs have to create new tailored models for ATs. Two managers added that current models do not provide sufficient technical details (IT-G, PublicInsureCo; CLBPA, RetailCo).

(2) Availability: ATs expect high availability of EAs. 37.50% of AT members said that they have difficulties in finding an EA, while 62.50% had a rather positive perception. 53.85% of all EAs reported that ATs have issues finding an EA, while the remaining 46.15% claimed that this is not the case. In contradiction, all managers felt that ATs have no difficulties finding EAs.

(3) Communication: Among all stakeholder groups, ATs were the least satisfied with the communication with EAs. They stated that EAs are overloaded (PO, CarCo) and lack technical know-how (SM, RetailCo), affecting the speed of communication (PO, CarCo). Further, AT members stated that direct communication channels are missing and that they need a lot of effort to identify contact persons (AT, RetailCo; AD, GlobalInsureCo). At PublicInsureCo, the perception of the communication was rather positive. An agile developer (AD) of PublicInsureCo explained that the required information is provided by EAs and that existing communication channels work. An AD of GlobalInsureCo also positively mentioned that EAs are perceived as valuable through direct contact and open dialogue. Similar to ATs, EAs also felt that the expectations regarding the communication are not satisfied. EAs argued that there are not able to communicate more intensively with ATs due to capacity restrictions (EA, CarCo; EA, RetailCo). They also recognized the issue of indirect communication with ATs via DAs, thus leading to dissatisfactions by ATs and their unwillingness to communicate with EAs (EA, PublicInsureCo). An EA of GlobalInsureCo explained the communication between EAs and ATs is good through the use of training and tools, but also noted that existing communication formats are not yet known by everyone and therefore sometimes communication is missing. An EA of CarCo had a more positive perception: "*There are no complaints*". Two EAs proposed one suggestion for improving this situation, namely, to clarify the roles and responsibilities of EAs so that the different expectations can be met (EA, RetailCo; EA, PublicInsureCo). Also, EAs should offer a variety of communication channels for faster accessibility (EA, GlobalInsureCo). In this respect, the external perception of managers differs considerably from that of ATs and EAs. The managers stated that EAs work closely with ATs (CTO, CarCo; CLAC, GlobalInsureCo), have sufficient time to collaborate (CTO, CarCo), and are always available to them (CLBPA, RetailCo).

(4) Involvement: The majority of ATs felt left out regarding their involvement in the architecture process because of weak collaboration (PO, CarCo; PO, ITCo; PO, SM, RetailCo). An PO of ITCo added: "*[ATs] are simply not asked*". Some interviewees called for greater involvement (PO, CarCo; SM, RetailCo). At PublicInsureCo, the opinion of one AD was very positive: "*Nothing is missing*". Similarly, an AD of GlobalInsureCo mentioned that teams are well involved in architecture decisions such as deciding on best practices, software architecture or code libraries. The self-perception of EAs is similar to that of ATs. The primary reason given was that the capacity of EAs is fully exhausted (EA, CarCo; EA, RetailCo; EA PublicInsureCo). According to one EA of PublicInsureCo, ATs do not necessarily feel to be involved in architecture processes due to the lack of support given by EAs. In addition, the involvement must be clarified in advance and optimized: "*Wherever there is a need, the way of involvement must be clarified*" (EA, PublicInsureCo). The feedback from ATs helps to optimize the capacities of EAs and thus enables their involvement in architecture processes (EA, RetailCo). Compared to other stakeholder groups, the managers' perception is very positive. At GlobalInsureCo, ATs are actively involved in architecture processes through architecture coordination circles (CLAC, GlobalInsureCo). With this respect, the CLBPA of RetailCo explained that the active involvement of ATs facilitates the creation of the to-be enterprise architecture. In contrast, the CTO of CarCo explained that not all ATs can be involved in architecture processes as this constitutes a scaling issue and they lack the necessary overarching view of the organization.

(5) Support: The majority of AT members are not satisfied with the support of EAs. They complained about the lack of support from EAs due to capacity problems, work overload, and a lack of technical know-how (PO, CarCo; PO, ITCo; SM, RetailCo; AD, PublicInsureCo). Few AT members were satisfied to some extent and reported positive feedback (PO, CarCo; SM, PO, RetailCo; AD, GlobalInsureCo). The added value of EAs was perceived positively in cases in which they provide high-level requirements (PO, CarCo) and offer top-down informative, consultative, and mediatory support (AD, GlobalInsureCo). The self-perception of EAs on this point is similar to that of ATs. The EAs mentioned time and capacity restrictions as main reasons for inadequate support (EA, CarCo; EA RetailCo; EA, GlobalInsureCo). Two EAs explained that architecture models provided are too unspecific and the expected support cannot be given (EA, CarCo; EA, RetailCo). In many cases, EAs do not have the necessary technical know-how to support ATs adequately (EA, CarCo). One EA of PublicInsureCo also stated that some ATs do not need support by EAs. Both EAs of GlobalInsureCo felt that ATs are satisfied as "*concrete added value is created in the teams*". Among all three stakeholder groups, the managers had the most positive perception. The CTO of CarCo praised that their EAs receive a lot of positive feedback and that competent EAs are highly demanded by ATs. He noted that EAs lack of capacity and thus need to focus on the most important projects with a high economic impact (CTO, CarCo). The CLBPA of RetailCo was very pleased with the support as "*they have a lot of educational and explanatory work to do*". The CLAC of GlobalInsureCo perceived the role of EAs as more active, because they "*join the team and are part of the team, not by e-mail, phone or other means*".

(6) Feedback: The AT members had a rather negative opinion since the feedback culture is not lived and there is a lack of feedback mechanisms between ATs and EAs (SM, RetailCo; AD, PublicInsureCo). According to an AD of GlobalInsureCo, EAs should show more initiative to gather feedback from ATs through observation and direct team collaboration. Due to the lack of capacity, EAs are not able to receive and process feedback (PO, ITCo). Only one AT member indicated that his feedback was well received and processed (PO, CarCo). The majority of EAs felt that the expectations of the ATs for the opportunity to provide feedback are met. They explained that ATs have sufficient feedback opportunities, e.g., workshops, e-mail or personal face-to-face meetings (EA, CarCo; EA, ITCo; EA, RetailCo; EA, PublicInsureCo; EA, GlobalInsureCo). One EA of CarCo stated: "*Wouldn't know how to improve it*". An EA of ITCo opposed since structured feedback mechanisms are not established or are further improvable. One EA of GlobalInsureCo added that some EAs do not respond to feedback of ATs at all. The perception of the managers is similar to that of EAs. Many ATs still perceive EAM as an ivory tower and therefore give little feedback to the EAs (CTO, CarCo). Existing feedback is driven top-down by EAs due to immature feedback culture and its dependency to the organizational structure (IT-G, PublicInsureCo; CLAC, GlobalInsureCo). One solution would be establishing stable feedback channels between ATs and EAs (CLAC, GlobalInsureCo).

(7) Recommendation: We used the Net Promoter Score (NPS) to obtain an overview of the satisfaction of all stakeholder groups. Here, we asked the question: *How likely is it that you would recommend an EA to an AT*? The participants were able to give an answer ranging from 0 (*not at all likely*) to 10 (*extremely likely*). Respondents with a score of 9 or 10 are called promoters. Interviewees answering a 7 or 8 are denoted as passives. Detractors are those with a score of 0 to 6. Finally, the NPS was calculated as the percentage of promoters minus the percentage of detractors. Figure 1 shows the NPS calculated for each stakeholder group as well as the overall NPS.

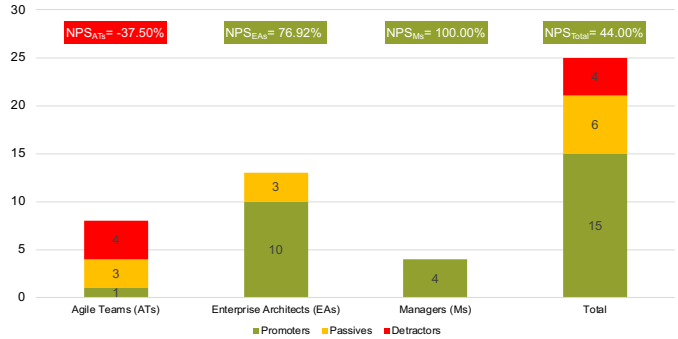


Figure 1. Overview of the NPS of all stakeholder groups (N=25)

Four interesting observations can be made: First, ATs do not seem to be satisfied with the current support by EAs (NPS = -37.50%). Second, most of the EAs would recommend themselves, thus seem to be satisfied

with how they support ATs (NPS = 76.92%). Third, all managers recommend EAs to ATs (NPS = 100.00%). Fourth, a comparison of all stakeholder groups reveals that ATs are least satisfied with the current situation and that the managers recognize the added value of EAs, confirming our previous observations and findings. The overall NPS value is 44.00%.

Figure 2 provides an aggregated overview of the respective ratings by each stakeholder group.

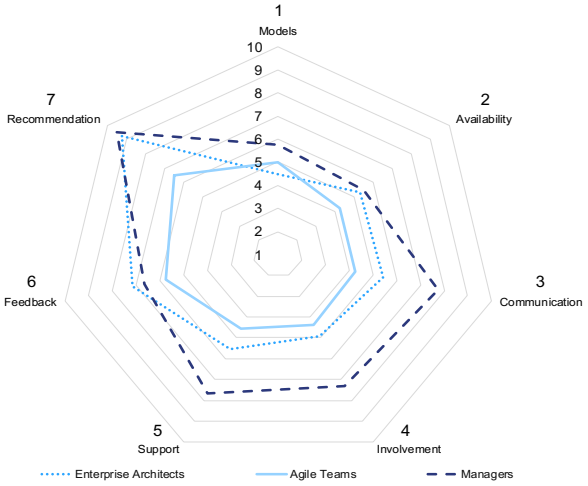


Figure 2. Overview of the rating on how different stakeholder groups perceive to what extent the ATs' expectations are fulfilled by EAs (N=25)

Discussion

Key Findings

Five key findings emerge from this multiple-case study: First, our results indicate that ATs mainly expect technological guidance and support from EAs. We are consonant with Drews et al. (2017) and believe that the role of the EA should be primarily technology focused. To this end, EAs must leave the "ivory tower" and keep their technical knowledge up-to-date (Kulak and Li 2017). Second, the lack of capacity and the high workload of EAs makes it difficult to deliver their services on time and in appropriate quality. Some EAM initiatives of the case organizations use agile and lean methods to address the above-mentioned problem. This finding is in line with Hanschke et al. (2015) and Hauder et al. (2014) that companies are applying agile and lean methods to increase the efficiency of EAM initiatives. Third, we made similar observations as Canat et al. (2018) that the communication between EAs and ATs is an issue. We observed that EAs work closely with ATs at the beginning of a project life-cycle and leave them after a project matured, similar to the way described in the Enterprise Unified Process (Ambler 2002). In addition, we witnessed that the communication between ATs and EAs via a third persons is considered suboptimal. Fourth, the fear of the "*Big Design Upfront*" becomes reality for ATs as they feel excluded from architecture processes. Although ATs follow the emergent design principle of the Agile Manifesto (Beck et al. 2001), we found that in many cases ATs ask for involvement in the creation of an intentional architecture. Fifth, the positive perception of managers and the rather negative attitude of ATs towards EAs show that the added value of EAs has not yet reached the team level, which is a similar result to that of (Canat et al. 2018).

Limitations

For the evaluation of possible validity threats of our observations, we used the assessment scheme recommended by (Runeson and Höst 2008). Since the case study is exploratory and does not seek to establish causal relationships, internal validity is not a concern. *Construct validity* is concerned with whether the study design represents a proper investigation of the research questions. We took three countermeasures to address this threat. First, we used several sources for data collection. These included semi-structured interviews with various stakeholder groups and internal documents of the cases. Second, the interviews were transcribed and coded by one researcher and then reviewed by a second researcher.

Third, key informants of the case organizations reviewed the interview results to establish a chain of evidence. *External validity* relates to the generalization of the findings and to what extent the results are of interest outside the investigated cases. For this purpose, we focused on the literal replication of our cases and aimed for analytical generalization by providing a thorough description of the cases. Our case study provides profound details so that this validity can be addressed by identifying similarities between other organizations and the characteristics of our cases. *Reliability* refers to whether the case study is conducted in a robust manner and whether a replication by other researchers would yield the same results, i.e., researcher bias. Three countermeasures were taken to counteract researcher bias. First, there were always two researchers present in the interviews. Second, the data analysis was checked for consistency and completeness by a second researcher. Third, all reports sent to the companies were revised by another researcher and discussed with company representatives.

Conclusion and Future Work

Confronted with the imperatives of competitive environments, firms are urged to undergo large-scale agile transformations to respond to environmental changes (Fuchs and Hess 2018; Gerster et al. 2018). The EAM function provides governance mechanisms to coordinate and align multiple large-scale agile activities to achieve desirable organization-wide effects and agility (Drews et al. 2017; Greefhorst and Proper 2011). The expectations of EAs and ATs are not always frictionless as both exhibit antithetic ways of thinking and mindsets (Kulak and Li 2017). Our research is motivated by the lack of empirical studies on the collaboration between EAs and ATs. We conducted an exploratory study with five cases to get an understanding of this field of tension.

Derived insights are threefold: First, communication between EAs and ATs is primarily driven by SAs, which increases information loss and misunderstanding. Second, all interviewees request a supporting EA role in the future, which is characterized by a deep understanding of technologies and broad communication skills. However, this new role requires EAs to cede decision-making power and possess sound reasoning skills. Third, the workload of EAs grew due to the multitude of ATs. Consequently, EAs have difficulties in handling all requests of ATs on architectural issues.

Finally, this article leaves some room for further studies. We encourage researchers to perform longitudinal studies on how the expectations of ATs for EAs vary throughout large-scale agile transformations. Researchers should also conduct cross-case analyses with the goal to compare how the role of the EA is realized in different types of organizations.

References

Ambler, S. 2002. *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*, New York: John Wiley & Sons.

Babar, M. A. 2009. "An Exploratory Study of Architectural Practices and Challenges in Using Agile Software Development Approaches," in *2009 Joint Working IEEE/IFIP Conference on Software Architecture European Conference on Software Architecture*, pp. 81–90.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., and others. 2001. *Manifesto for Agile Software Development*.

Bente, S., Bombosch, U., and Langade, S. 2012. "Collaborative Enterprise Architecture," *Collaborative Enterprise Architecture*, Burlington: Morgan Kaufmann.

Besson, P., and Rowe, F. 2012. "Strategizing Information Systems-Enabled Organizational Transformation: A Transdisciplinary Review and New Directions," *The Journal of Strategic Information Systems* (21:2), pp. 103–124.

Canat, M., Català, N. P., Jourkovski, A., Petrov, S., Wellme, M., and Lagerström, R. 2018. "Enterprise Architecture and Agile Development - Friends or Foes?," in *22nd International Enterprise Distributed Object Computing Workshop*.

Dikert, K., Paasivaara, M., and Lassenius, C. 2016. "Challenges and Success Factors for Large-Scale Agile Transformations: A Systematic Literature Review," *Journal of Systems and Software* (119), pp. 87–108.

Dingsøyr, T., Fægri, T. E., and Itkonen, J. 2014. "What Is Large in Large-Scale? A Taxonomy of Scale for Agile Software Development," in *Product-Focused Software Process Improvement*, A. Jedlitschka, P.

Kuvaja, M. Kuhrmann, T. Männistö, J. Münch, and M. Raatikainen (eds.), Cham: Springer International Publishing, pp. 273–276.
Dingsøyr, T., Moe, N., Fægri, T., and Seim, E. 2018. "Exploring Software Development at the Very Large-Scale: A Revelatory Case Study and Research Agenda for Agile Method Adaptation," *Empirical Software Engineering* (23:1), pp. 490–520.
Drews, P., Schirmer, I., Horlach, B., and Tekaat, C. 2017. "Bimodal Enterprise Architecture Management: The Emergence of a New EAM Function for a BizDevOps-Based Fast IT," in *21st International Enterprise Distributed Object Computing Workshop*, pp. 57–64.
Easterbrook, S., Singer, J., Storey, M.-A., and Damian, D. 2008. "Selecting Empirical Methods for Software Engineering Research," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. Sjøberg (eds.), London: Springer, pp. 285–311.
Fuchs, C., and Hess, T. 2018. "Becoming Agile in the Digital Transformation: The Process of a Large-Scale Agile Transformation," in *39th International Conference On Information Systems*, San Francisco.
Gerster, D., Dremel, C., and Kelker, P. 2018. "'Agile Meets Non-Agile': Implications of Adopting Agile Practices at Enterprises," in *24th Americas Conference on Information Systems*.
Greefhorst, D., and Proper, E. 2011. *Architecture Principles: The Cornerstones of Enterprise Architecture*, Berlin: Springer.
Gregory, P., Barroca, L., Taylor, K., Salah, D., and Sharp, H. 2015. "Agile Challenges in Practice: A Thematic Analysis," in *International Conference on Agile Software Development*, pp. 64–80.
Hanschke, S., Ernsting, J., and Kuchen, H. 2015. "Integrating Agile Software Development and Enterprise Architecture Management," in *48th Hawaii International Conference on System Sciences*.
Hauder, M., Roth, S., Schulz, C., and Matthes, F. 2014. "Agile Enterprise Architecture Management: An Analysis on the Application of Agile Principles," in *International Symposium on Business Modeling and Software Design*.
Hekkala, R., Stein, M.-K., Rossi, M., and Smolander, K. 2017. "Challenges in Transitioning to an Agile Way of Working," in *50th Hawaii International Conference on System Sciences*, pp. 5869–5878.
Kettunen, P., and Laanti, M. 2017. "Future Software Organizations - Agile Goals and Roles," *European Journal of Futures Research* (5:1), p. 16.
Kulak, D., and Li, H. 2017. "The Journey to Enterprise Agility: Systems Thinking and Organizational Legacy," *The Journey to Enterprise Agility: Systems Thinking and Organizational Legacy*, Cham: Springer International Publishing.
Lethbridge, T. C., Sim, S. E., and Singer, J. 2005. "Studying Software Engineers: Data Collection Techniques for Software Field Studies," *Empirical Software Engineering* (10:3), pp. 311–341.
Miles, M., Hubermann, M., and Saldana, J. 2014. *Qualitative Data Analysis: A Methods Sourcebook*, Thousand Oaks: Sage Publications.
Rolland, K. H., Fitzgerald, B., Dingsøyr, T., and Stol, K.-J. 2016. "Problematizing Agile in the Large: Alternative Assumptions for Large-Scale Agile Development," in *37th International Conference on Information Systems*.
Rouhani, B. D., Mahrin, M. N., Nikpay, F., Ahmad, R. B., and Nikfard, P. 2015. "A Systematic Literature Review on Enterprise Architecture Implementation Methodologies," *Information and Software Technology* (62), pp. 1–20.
Runeson, P., and Höst, M. 2008. "Guidelines for Conducting and Reporting Case Study Research in Software Engineering," *Empirical Software Engineering* (14:2), p. 131.
Scheerer, A., Hildenbrand, T., and Kude, T. 2014. "Coordination in Large-Scale Agile Software Development: A Multiteam Systems Perspective," in *47th Hawaii International Conference on System Sciences*, pp. 4780–4788.
Stake, R. 1995. *The Art of Case Study Research*, Thousand Oaks: Sage Publications.
Theuerkorn, F. 2004. *Lightweight Enterprise Architectures*, Boca Raton: Auerbach Publications.
Uludağ, Ö., Kleehaus, M., Xu, X., and Matthes, F. 2017. "Investigating the Role of Architects in Scaling Agile Frameworks," in *21st International Conference on Enterprise Distributed Object Computing Conference*, pp. 123–132.
Wilkinson, M. 2006. "Designing an 'Adaptive' Enterprise Architecture," *BT Technology Journal* (24:4), pp. 81–92.
Winter, R. 2016. "Establishing 'Architectural Thinking' in Organizations," in *The Practice of Enterprise Modeling*, J. Horkoff, M. A. Jeusfeld, and A. Persson (eds.), Cham: Springer International Publishing, pp. 3–8.
Yin, R. 1994. *Case Study Research: Design and Methods (5th Ed.)*, Thousand Oaks: Sage Publications.

Investigating the Role of Enterprise Architects in Supporting Large-Scale Agile Transformations: A Multiple-Case Study

Completed Research

Ömer Uludağ
Technische Universität München
oemer.uludag@tum.de

Florian Matthes
Technische Universität München
matthes@tum.de

Abstract

In today's competitive environments, companies must cope with changing customer demands, regulatory uncertainties, and new technological advances. To this end, companies increasingly undergo large-scale agile transformations to meet these requirements. In recent years, enterprise architecture management has established itself as a valuable governance mechanism for coordinating large-scale agile transformations by connecting strategic considerations to the execution of transformation projects. Empirical studies investigating the role of enterprise architects (EAs) in this context are still scarce. We present a multiple-case study of five major German companies that aims to shed light on the role of EAs in supporting large-scale agile transformations. Based on our results from eighteen interviews, we present a set of typical responsibilities of EAs. We also describe the expectations of various stakeholders towards EAs and the challenges they face.

Keywords

Enterprise architecture management, large-scale agile transformation, multiple-case study

Introduction

Today's competitive environments are characterized by uncertainty and turbulence resulting from regulatory uncertainties, changing customer demands, and new technological advances (Kettunen and Laanti 2017; Sherehiy et al. 2007). As a result, companies are urged to undergo organizational transformations to respond to dynamic environments and to sustain their survival (Besson and Rowe 2012; Gerster et al. 2018). To master their respective digital transformations, organizations strive to become agile (Gerster et al. 2018; Rogers 2016). The extensive introduction of agile methods is a common approach to address relevant issues of an organizational digital transformation (Bharadwaj et al. 2013; Gerster et al. 2018). The large-scale adoption of agile methods in organizations often leads to large-scale agile transformations as part of their organizational digital transformations (Dikert et al. 2016; Fuchs and Hess 2018). Thereby, the term "*large-scale agile transformation*" can be defined as the transition from traditional development approaches to agile practices in large multi-team or enterprise-wide settings (Gerster et al. 2018). A large-scale agile transformation may involve a one-time big bang transfer to agile methods or a step-wise approach where an agile pilot is scaled up into a large setting (Fuchs and Hess 2018). Large-scale agile transformations entail new managerial challenges such as skepticism towards the new way of working (Dikert et al. 2016), coordination and alignment issues between large-scale agile endeavors as well as between agile and non-agile teams (Scheerer et al. 2014), and tensions between top-down architecture governance and bottom-up autonomy of agile teams (ATs) (Kulak and Li 2017). In recent years, enterprise architecture management (EAM) has established itself as a valuable governance mechanism for coordinating enterprise transformations by connecting strategic considerations to the execution of large agile projects (Greefhorst and Proper 2011; Hauder et al. 2014). In large-scale agile transformations, the role of EAM is also changing from enforcing technological standards to advising ATs on their architectural decisions (Hanschke et al. 2015). Therefore, the new role of EAM focuses mainly on cross-team issues with harmonizing governance efforts across ATs and guiding them through

architectural roadmaps (Uludağ et al. 2017). In the context of large-scale agile transformations, there are two important architecture roles: first, enterprise architects (EAs) can support large-scale agile transformations by resolving technical dependencies of ATs on a portfolio level and shaping the overall strategic vision of the organization (Horlach et al. 2020; Uludağ et al. 2017). Second, software architects can support large-scale agile transformations on the program or team level by taking architecture decisions for ATs (Horlach et al. 2020; Uludağ et al. 2017). Notwithstanding the importance of EAs in supporting large-scale agile transformations, only a few articles are available that describe the roles of EAs in this context (cf. (Canat et al. 2018; Horlach et al. 2020)). None of these articles provide a cross-case analysis of how EAs can support large-scale agile transformations and the challenges they face when collaborating with agile teams. We aim to fill this gap by raising the following research questions (RQs):

RQ1: Which responsibilities do enterprise architects have in large-scale agile transformations?

RQ2: How should enterprise architects support large-scale agile transformations?

RQ3: Which challenges do enterprise architects face in supporting large-scale agile transformations?

Background and Related Work

Architects in Agile Development

Agile methods such as Scrum and Extreme Programming, which more or less adhere to the values of the Agile Manifesto (Beck et al. 2001), assume that the best architectures, requirements, and design emerge from self-organizing teams (Abrahamsson et al. 2010). They do not provide detailed guidance on building a software architecture or on the role of software architects. The basic idea of these methods is that architecture is a natural outcome of sprints and re-factorings (Leffingwell 2007). Despite differing views on the compatibility of agility and architecture, the interests of scientists and practitioners aiming to combine both concepts is increasing (Babar 2009). Abrahamsson et al. (2010) argue that the best solution is to find a balance between these two extreme approaches and recommend the integration of architects into agile projects. Angelov et al. (2016) propose three scenarios for the integration of architects[1] in agile projects: *internal architect*, *external architect*, and *internal & external architect*. The first scenario describes different types of internal architects. The *architecture agent* is usually a senior developer who acts as an architect. Based on his expertise, he supports the team by taking responsibility of architectural decisions (Angelov et al. 2016; Fowler 2003). There can also be a dedicated *supporting architect* role in the team, which mentors and consults the team in architectural topics. In addition, an AT can also have an assigned *classical architect* who makes all important architectural decisions for the team. If there is no expert, the entire team works as an internal architect. In this case, there is *no dedicated architect* in the team (Angelov et al. 2016; Toth 2015). The second scenario describes the role of an *external architect*. This architect is characterized by a supporting and guiding function (Toth 2015). He is assigned to multiple teams and collaborates with other architects (Angelov et al. 2016). The external architect keeps an eye on the big picture of the architecture. This setup gives the teams the support they need to integrate the architecture into their teams (Angelov et al. 2016; Fowler 2003). The external architect can be either an architect who supports or guides ATs (Toth 2015). In the third scenario, there is both an *internal & external architect*. The internal architect focuses on inter-team communication, manages the current code, and eliminates architectural problems that occur within the team. The external architect takes care of the overall architecture, makes decisions to meet the architectural requirements, and acts as an external coordinator between teams (Angelov et al. 2016; Toth 2015).

Enterprise Architects in Large-Scale Agile Development

Based on ten expert interviews with three German companies, Hanschke et al. (2015) describe how agile methods can be used to create EA deliverables and how EAs can collaborate with ATs. Based on twelve interviews with four companies, Canat et al. (2018) reveal that their cases do not regard agile and architecture as opposites. The authors also claim that the extensive use of modeling at the team level is

[1] The findings of Angelov et al. (2016) refer to the involvement of architects in agile projects in general, including both enterprise and software architects. Thus, the three scenarios for the integration of architects in agile projects apply to both architect roles.

considered as over-architecting as the lower levels are more dynamic. They also describe that EAs and ATs have communication gaps and that EAs have no difficulties in working with AT, but that ATs tend to disagree with the working methods of EAs. Horlach et al. (2020) propose six design principles for how agile teams can establish architectural thinking so that they can see the 'big picture' of an organization in terms of its strategic objectives and IT landscape to avoid organizational inertia or technical debts.

Research Methodology

Being induced by a practical problem, we apply case study research since it gives an in-depth overview of contemporary phenomena (Yin 1994).

Case Study Design: We aim to explore the role of EAs in supporting large-scale agile transformations. Based on this goal, we formulated three research questions. We use a case study design with five companies (Yin 1994). The cases were purposefully selected as they undergo major transformations and their EAs face new challenges while working with ATs. We selected cases from different industries to avoid industry bias. Table 1 presents an overview of the case companies and interviews.

Code name	Employees	Interviews	Position of interviewees
Global insurance company ("GlobalInsureCo")	140,000+	4	Agile developer; enterprise architect; chapter lead agile coaching
Car manufacturer ("CarCo")	130,000+	4	Chief technology officer; enterprise architect; requirements engineer; scrum master
Information technology company ("ITCo")	7,000+	2	Enterprise architect; product owner
Retail company ("RetailCo")	50,000+	4	Chapter lead business process architecture; enterprise architect; product owner; scrum master
Public sector insurance company ("PublicInsureCo")	6,700+	4	Agile developer; enterprise architect; head of IT governance department

Table 1. Overview of case organizations and conducted interviews

Data Collection: We focused mainly on first- and third-degree data collection techniques (Lethbridge et al. 2005). Using first-degree methods, we established direct contact with the subjects and collected data in real time. For this purpose, we conducted twelve individual and six group interviews. In nearly all cases, at least one senior executive, one EA, and one member of an AT were interviewed to receive a multifaceted perspective on the subject under investigation and to triangulate our results. The interviews followed a semi-structured questionnaire and were conversational to enable interviewees to describe their experiences and views in detail (Yin 1994). The interviews were mainly conducted in face-to-face meetings. At least two researchers were present in the interviews to strengthen observer triangulation (Runeson and Höst 2009). We enriched our interview findings with third-degree data collection techniques that supported us to analyze existing artifacts and available data. Here, slide decks and wiki pages of the cases gave us detailed information about their EAM initiatives. The selection of multiple data sources and roles from different case companies facilitated the triangulation of data sources (Stake 1995).

Data Analysis: The interviews were transcribed and coded with open coding (Miles et al. 2014). After initial coding, we consolidated preliminary codes and checked their consistency and completeness. We combined groups of code phrases into concepts that were linked to our research questions.

Results

Responsibilities of Enterprise Architects

We used exploratory questions to reveal the roles of EAs. Thus, we asked the respondents on the following categories: (1) responsibilities, (2) changes in function, (3) and shifts in working methods. First, we asked the interviewees to specify the responsibilities of EAs in large-scale agile transformations (see Table 2[2]).

[2] A "✓" indicates that a particular responsibility has been identified in a case while an "✗" means that a specific responsibility is not given.

No.	Responsibilities of enterprise architects	Global-InsureCo	CarCo	ITCo	Retail-Co	Public-InsureCo
#1	Collaborating with stakeholders to develop the company's architectural roadmap	✓	✓	✓	✓	✓
#2	Collaborating with ATs to guide them through architectural requirements and roadmaps	✓	✓	✓	✓	✓
#3	Facilitating architectural decision-making process of ATs	✓	✓	✓	✓	✓
#4	Creating and communicating architecture principles and ensuring their compliance	✓	✓	✓	✓	✓
#5	Creating strategic technical directions of the company	✓	✓	✓	✓	✓
#6	Organizing communities of practices for architecture and architecture boards	✓	✓	✓	✓	✓
#7	Driving architectural decisions of the software products	✗	✓	✓	✓	✗
#8	Identifying and managing dependencies between ATs	✓	✗	✓	✗	✓
#9	Ensuring the reuse of enterprise assets	✓	✓	✓	✗	✗
#10	Mentoring and coaching ATs and other architects with architectural basics and best practices	✓	✗	✗	✓	✓
#11	Consulting enterprise executives	✗	✓	✗	✓	✓
#12	Fostering technical excellence	✓	✗	✗	✓	✓
#13	Discovering and experimenting with new technologies	✓	✓	✗	✗	✓
#14	Creating architectural blueprints and standards	✓	✓	✗	✓	✗
#15	Consulting ATs for architectural problems	✓	✓	✗	✗	✓
#16	Defining a target architecture of the software products	✗	✓	✗	✓	✓
#17	Defining an architectural solution space for ATs	✓	✓	✗	✓	✗

Table 2. Responsibilities of enterprise architects in large-scale agile transformations

We observed several similarities between the case companies regarding the responsibilities of EAs. First, their primary responsibility was not just to create architectural artifacts, such as standards or principles, but also to closely communicate and collaborate with various stakeholders, in particular with ATs. Second, their task included the creation of a common architectural understanding within the organization. For instance, they provided training courses to developers or architects regarding architecture basics. In addition, they organized communities to facilitate the exchange of information and knowledge. Third, EAs were also responsible for steering and coordinating large-scale agile endeavors by creating technology and business roadmaps and defining target architectures. In all cases, the ongoing transformations had a great impact on the role of EAs leading to a number of changes in their responsibilities. At GlobalInsureCo, the architectural decision-making authority was shifted from EAs to ATs. First, ATs were no longer dependent on the approval of each architectural decision by the EAs but were empowered to make their own decisions and were also accountable for them. Second, EAs were responsible for creating and communicating architectural guidelines to ATs. Both changes were also reflected by an AT member:

> "*The decision-making power of EAs have been considerably reduced as the teams themselves are now responsible for their architecture. Now, their responsibility is to explain architectural requirements to create context and achieve team acceptance.*" *— Agile Developer, GlobalInsureCo*

EAs were also regarded as service providers offering "*enterprise architecture as a service*", e.g., providing tools or offering consultation for architectural issues. Similar to GlobalInsureCo, CarCo's EAs had a supportive and consulting role by assisting ATs for resolving architectural impediments. In addition, EAs no longer defined an upfront architecture for software products but instead enabled a collaborative process to create target architectures together with ATs. The organization also underwent a paradigm shift away from monolithic architectures to microservice architectures, requiring EAs to support several ATs. This change was accompanied by an excessive workload of EAs:

> "*Enterprise architects are now overloaded. Previously, they had to accompany 4 system teams, now they have to assist more than 20 component teams.*" *— Scrum Master, CarCo*

EAs recognized this issue and stated that their task is to convey "*enterprise architecture as a skill*" to ATs. At ITCo, the main change was seen in providing context information to ATs and identifying dependencies to other teams. EAs at RetailCo were required to delegate responsibilities to ATs. As a result, EAs had to work more closely with ATs and to convince them argumentatively to move ATs in the right direction to

achieve the overall architecture strategy. EAs of PublicInsureCo were affected by the ongoing transformations and gained an increasingly consultative role and performed fewer governance control functions. To this end, EAs lost their veto right in architectural decisions made by ATs. They used principles to provide an orientation for ATs without dictating them how to realize the specified principles.

We were also curious about the changes in their working methods. Table 3 provides an overview of the changes. Again, a few similarities between the cases could be observed. First of all, EAs had increasingly adopted agile and lean practices to avoid long conceptualization phases or the creation of outdated architecture models. Second, EAs spent most of their time to communicate and collaborate with ATs, e.g., providing training courses or organizing community events. Third, they also built more technical skills to provide better technological assistance.

No.	Changes in the working methodology of enterprise architects	Global-InsureCo	CarCo	ITCo	Retail-Co	Public-InsureCo
#1	Avoiding big design upfront by creating the simplest architecture that can possibly work	✗	✓	✓	✓	✓
#2	Leaving the 'ivory tower' by closely collaborating with ATs	✓	✓	✗	✓	✓
#3	Involving ATs in the architectural decision-making process instead of deciding over their heads	✓	✓	✗	✓	✓
#4	Using new tools such as collaboration tools of ATs	✓	✓	✗	✓	✓
#5	Creating architecture roadmaps with shorter conceptualization phases and observation periods	✗	✗	✓	✓	✓
#6	Using also agile and lean practices such as sprints, backlogs, and kanban boards	✓	✓	✗	✗	✓
#7	Organizing training courses and workshops for ATs	✓	✗	✗	✓	✓
#8	Building technical skills and capabilities	✓	✗	✗	✓	✓
#9	Conducting architecture spikes to explore new technologies	✗	✓	✓	✗	✓
#10	Aligning architectural decisions via communities rather than via architecture boards	✓	✗	✗	✓	✗
#11	Specifying solution spaces rather than creating restrictive requirements limiting the design freedom of ATs	✓	✓	✗	✓	✗
#12	Preferring communication with ATs based on personal conservations over communication based on documents	✓	✗	✗	✓	✓
#13	Attending events of ATs such as plannings or demos	✗	✗	✗	✓	✓
#14	Preferring architectural decisions based on consensus	✓	✗	✗	✓	✗
#15	Coding with ATs rather than being a 'PowerPoint' architect	✓	✗	✗	✓	✗

Table 3. Changes in the working methodologies of enterprise architects

At RetailCo, some EAs were treated as developers and were therefore perceived as part of ATs. Consequently, they were no longer part of the traditional governance cluster, but rather part of the product organization. EAs at PublicInsureCo also switched from phased support of ATs to more continuous support which is why they accompanied projects longer than before. Moreover, they closely collaborated with domain and software architects within architecture communities. They were also regularly invited to Scrum events of ATs.

Supporting Role of Enterprise Architects

We asked the interviewees to describe how EAs should support large-scale agile transformations. Table 4 gives an overview of the identified expectations. According to our findings, the most salient expectations were that EAs should align ATs to work on the business visions and to develop a fundamental understanding of architecture so that ATs can work on achieving the company's objectives themselves.

No.	Expectations for enterprise architects	Global-InsureCo	CarCo	ITCo	Retail-Co	Public-InsureCo
#1	Demonstrating the value of architecture	✓	✓	✓	✓	✓
#2	Aligning ATs with enterprise goals through principles	✓	✓	✓	✓	✓
#3	Fostering a common understanding for architecture	✓	✓	✗	✓	✓
#4	Entrusting more architecture responsibilities to ATs	✓	✗	✗	✗	✓
#5	Implementing architecture blueprints	✓	✗	✗	✗	✓
#6	Defining and communicating architecture standards	✓	✓	✗	✗	✗

#7	Supporting in designing team structures	✓	✓	✗	✗	✗
#8	Developing a continuous delivery pipeline for ATs	✓	✗	✗	✓	✗
#9	Specifying strategic goals for ATs	✗	✓	✓	✗	✗
#10	Identifying dependencies between ATs	✗	✓	✓	✗	✗
#11	Converting abstract themes into concrete initiatives	✗	✓	✓	✗	✗
#12	Consulting ATs regarding tool stacks	✗	✓	✗	✗	✓
#13	Establishing an alignment between business and IT	✗	✓	✗	✗	✓

Table 4. Expectations for enterprise architects to support large-scale agile transformations

Second, we were curious about the types of EAs exercised in the cases (see Table 5).

Type \ Case	Global-InsureCo	CarCo	ITCo	Retail-Co	Public-InsureCo
No dedicated architect	✗	✗	✗	✗	✗
Architecture agent	✓	✗	✓	✗	✓
Supporting architect	✓	✓	✓	✓	✓
Classic architect	✓	✓	✓	✓	✓

Table 5. Types of enterprise architects exercised in the case organizations

Two interesting observations can be made: First, none of the cases had ATs taking over the responsibilities of EAs as a whole team. Rather, the responsibilities of EAs were either exercised by a dedicated EA or taken over by an experienced developer acting as an architecture agent. Second, all cases had EAs acting either supportive or prescriptive. For instance, at GlobalInsureCo, architecture agents were regular AT members developing software and making architectural decisions. Although they had detailed knowledge about the software, interviewees had a rather negative attitude towards this type of architect. Respondents argued that architecture agents lacked an overarching perspective on the organization and comprehensive knowledge of enterprise architecture. Moreover, one interviewee stated that the architecture solutions developed by architecture agents were less sustainable because they were usually produced quickly. The role of the architecture agent role was also seen as less attractive due to the dual role in the team and additional effort it entailed:

"The role of the architecture agent is the one you least want to have." — Enterprise Architect, ITCo

In all cases, supporting EAs were primarily characterized by their proximity to ATs. However, we observed three scenarios of how they supported ATs. In the first scenario, they were requested and used as "*ad-hoc fire extinguishers*" when ATs did not know how to progress. In the second scenario, they joined ATs so that the teams could use their skills to solve urgent architectural issues. In the third scenario, EAs accompanied several ATs as mentors who were available for advice and support. Moreover, supporting EAs were characterized by comprehensive technical know-how with solid knowledge of tool stacks and programming languages. In all cases, supporting EAs had less decision-making authority than classic architects. Thus, they had to argumentatively convince ATs to carry out architecture activities and motivate them to do so. Advocates of the supporting EA argued that they were much closer to the needs of ATs regarding architecture. They claimed that supporting EAs achieved greater effectiveness and buy-in among ATs by increasing their intrinsic motivation. A further advantage mentioned was that supporting EAs provided developers enough design freedom:

"The supporting enterprise architect facilitates the self-organization of agile teams."
— Head of IT Governance, PublicInsureCo

The interviewees also saw some drawbacks of the supporting EA role. First and foremost, their missing decision-making power was leading to a lack of escalation mechanisms in case of non-compliance with architectural recommendations. Along with this, an interviewee mentioned the possibility of greater frustration due to a lack of decision-making power and little assertiveness. The role was perceived very demanding as greater efforts were required to persuade ATs, especially when they did not adhere to guidelines. Moreover, the respondents stated that demonstrating the added value of architecture to ATs could be very challenging. One AT member claimed that the ability of the supporting EA to have an overview of the overall picture could diminish through greater involvement in agile projects. One respondent also pointed out the missing consequences in case of bad consultations:

"*Consultation without responsibilities and obligations.*" — *Requirements Engineer, CarCo*

The classic EA was distinguished above all by his comprehensive enterprise architecture know-how and his focus on strategic themes. We observed that classic EAs were partly external architects making architectural decisions for ATs and taking responsibility for them. However, they worked more on higher management levels and were mainly involved in critical agile projects. In addition, classic EAs had a greater distance to ATs than supporting EAs. Here, several participants mentioned that classic EAs tended to work in the "*ivory tower*" and governed ATs by defining architectural requirements that they had to meet. The interviewees mentioned the decision-making authority of the classic EA as an advantage in moving ATs in the right architectural direction. In addition, classic EAs had escalation mechanisms in case of non-compliance with architecture guidelines. Classic EAs also had a good overview of the overall strategy and big picture of the organization. Due to the great distance to ATs, interviewees mentioned that classic EAs made decisions over the heads of ATs without knowing their actual concerns. ATs had greater difficulties in reaching classic EAs and providing feedback. Thus, classic EAs often encountered acceptance issues when working with ATs. We asked the interviewees to indicate which type of architect is desired in the future. The interviewees favored the supporting and classic EAs (see Figure 1). Our findings confirm that the role of the architecture agent is perceived unappealing for ATs and that they actually prefer supporting EAs. Different stakeholder groups wanted to have clear responsibilities in relation to enterprise architecture. Some respondents also indicated that both classic and supporting EAs should be exercised in the future, as supporting EAs would provide concrete technical guidance for ATs, while classic EAs would have a better overview of the macro level. Some interviewees suggested that supporting EAs should be equipped with decision-making authority to obtain a mix of classic and supporting EAs.

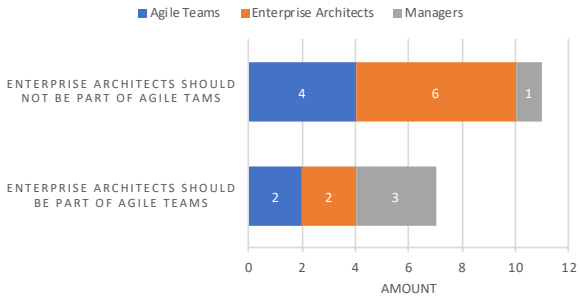


Figure 1. Types of enterprise architects that should be exercised in the future (n=23)

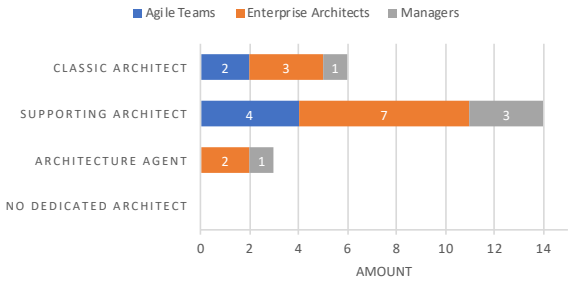


Figure 2. Involvement of enterprise architects as agile team members (n=18)

We were also interested in finding out whether EAs should be internal architects and part of ATs or whether they should be external architects supporting multiple ATs (see Figure 2). The majority favored external architects due to two reasons. First, they felt that ATs need some freedom to work and EAs should maintain their overarching view of the organization. Second, they argued that ATs demand clear guidance and support as needed. Some interviewees indicated that internal EAs would not be feasible due to time and capacity constraints. In contrast, some interviewees argued that EAs should be phase-based part of ATs, e.g., at the beginning of a project or depending on the architectural workload. Although the majority of ATs and EAs chose external architects, most of the managers voted for internal EAs. They justified this by saying that EAs should gain a better understanding of the current concerns of ATs:

"*Then they [enterprise architects] know which problems actually exist in the teams.*"
— *Chapter Lead Business Process Architecture, RetailCo*

Challenges faced by Enterprise Architects

We were curious about the problems EAs faced in supporting large-scale agile transformations. Table 6 provides an overview of the identified challenges. In all cases, EAs were confronted with ATs that did not understand or see the added value of enterprise architecture. Thus, ATs were slightly skeptical about external recommendations or guidelines from EAs. To this end, EAs without decision-making authority had difficulties in driving the organization's architecture strategy. Moreover, further challenges originated from the tension between agility and architecture. For instance, ATs were put under pressure to deliver business functionality, while long-term architectural improvements were often neglected. In contrast, EAs

had the mandate to build sustainable architectures reflecting the long-term strategy of the organization. This tension also led to the accumulation of technical debts. EAs had to deal with a high frequency of changes in IT systems as the architectural work of large-scale agile projects was more continuous. Due to a lack of capacity and appropriate tools, EAs struggled to track whether ATs were on the right path to achieve architecture goals. The ongoing agile transformations made the roles and responsibilities in architectural decisions for ATs and EAs increasingly unclear.

No.	Challenges faced by enterprise architects	Global-Insure-Co	CarCo	ITCo	Retail-Co	Public-Insure-Co
#1	Dealing with a lacking understanding of enterprise architecture	✓	✓	✓	✓	✓
#2	Balancing short-term and long-term planning	✗	✓	✓	✓	✓
#3	Balancing upfront and emergent architecture	✗	✓	✓	✓	✓
#4	Dealing with acceptance issues by ATs	✓	✗	✓	✓	✓
#5	Dealing with loss of decision-making power	✓	✗	✗	✓	✓
#6	Defining clear roles and responsibilities regarding architecture	✓	✓	✗	✓	✗
#7	Ensuring the adherence of ATs with architecture requirements	✓	✗	✗	✓	✓
#8	Creating proper upfront architecture of the systems	✗	✗	✓	✓	✗
#9	Dealing with high frequency of architectural decisions	✗	✓	✓	✗	✗
#10	Creating a comprehensive overview of application landscape	✗	✓	✗	✗	✓
#11	Dealing with technical implementations and respective technology trends	✗	✓	✗	✗	✓
#12	Dealing with turbulent environments	✓	✓	✗	✗	✗
#13	Establishing a lightweight review process for adapting new technologies	✓	✓	✗	✗	✗
#14	Balancing architecture improvements and business value	✗	✓	✗	✓	✗
#15	Managing technical debts	✗	✓	✓	✗	✗
#16	Providing ATs appropriate architecture artifacts	✗	✗	✓	✓	✗
#17	Tracking the progress of ATs in meeting architectural requirements	✗	✓	✗	✓	✗

Table 6. Challenges enterprise architects face in large-scale agile transformations

Discussion

Key Findings

Five key findings emerge from this multiple-case study: First, in line with Babar et al. (2013), we observed that ATs were increasingly empowered to make local architectural decisions leading to a shift of architectural decision responsibilities from EAs to ATs. It entailed a notable paradigm shift of mindsets, i.e., EAs had more of an "*advisory servant*" role rather than a "*command and control*" philosophy. Similar to Drews et al. (2017), we observed that EAs consulted and assisted ATs in the realization of architectural standards and acted as a facilitator for cross-team architecture exchange. Second, as "*agile breaks everything*" (Kulak and Li 2017), the large-scale adoption of agile methods had far-reaching implications on the working methods of EAs. The primary change we observed was that EAs no longer had big conception phases aiming to avoid "*Big Design Upfront*". Instead, the architectural work by EAs was more continuous and synchronized with the sprints of ATs. We agree with the opinion that "*agile and architecture can coexist as complementary approaches*" (cf. (Abrahamsson et al. 2010)) and believe that it not only exists between software architecture and agile development but also between enterprise architecture and agile development. Third, while the majority of the EAM frameworks and literature mainly focus on architectural artifacts in the form of models, principles or specifications, we realized that ATs demand more hands-on and technical solutions by EAs. For instance, they expect EAs to have dealt with architectural problems not only at the conceptual level but also at the technical level, i.e., developing small prototypes to understand the "*real concerns*" of ATs. EAs can meet this expectation by performing architecture spikes. Fourth, although the self-organization of ATs is a major success factor for large-scale agile endeavors (Dikert et al. 2016), we observed that they do not insist on full autonomy regarding architecture as long they are involved in architectural decision-making processes. Most participants believed that EAs are important in ensuring that ATs are on the right track. Fifth, Uludağ et al. (2019)

noted that internal software architects play a dual role in their teams by making architectural decisions and taking responsibility for them (classical architect) and helping the team in implementing the architectural decisions made (supporting architect). We came to similar results as Uludağ et al. (2019) only with the exception that EAs should not be part of ATs. Rather, ATs want to have external EAs that become part of ATs only for short periods of time.

Limitations

We discuss potential validity threats along with an evaluation scheme suggested by Runeson and Höst (2009). *Construct validity* reflects what extent the operational measures studied represent what the researcher has in mind (Runeson and Höst 2009). As a countermeasure, we interviewed several people with different roles. The interviews were also transcribed and coded by one researcher and then reviewed by a second researcher. Moreover, key informants of the cases reviewed the interview results to establish a chain of evidence. *Internal validity* is not a concern since this study is neither exploratory nor causal. *External validity* refers to the generalizability of the results (Runeson and Höst 2009). Thus, we focused on the literal replication of our cases and their analytical generalization. *Reliability* concerns to what extent data and analysis are dependent on the specific researcher (Runeson and Höst 2009). To counter this, two researchers were always present during the interviews. All reports sent to the companies were also revised by another researcher and reviewed by case representatives. Also, a case study database was created comprising case study documents such as audio recordings, interview protocols, and slide decks.

Conclusion and Future Work

Confronted with the imperatives of competitive environments, organizations undergo large-scale agile transformations to respond to environmental changes (Fuchs and Hess 2018; Gerster et al. 2018). EAs can support these transformations by aligning the individual project strategies of ATs with enterprise objectives and guiding ATs through business and technical roadmaps (Uludağ et al. 2017). Our research was motivated by the lack of empirical studies on the role of EAs in supporting large-scale agile transformations. We conducted a study with five cases to get an in-depth understanding of the responsibilities of EAs and the challenges they face. As future work, we encourage other researchers to conduct cross-case analyses and to compare digital natives with traditional companies that may have different collaboration cultures between EAs and ATs. This may help to get a better understanding of how the roles and responsibilities of EAs change in specific organizational environments.

In future work, researchers should perform explanatory studies to reveal potential relationships between exercised EA types and the realization of formulated EAM strategies. Researchers should also conduct case studies to identify best practices for non-decisive EAs to drive architectural initiatives.

References

Abrahamsson, P., Babar, M. A., and Kruchten, P. 2010. "Agility and Architecture: Can They Coexist?," *IEEE Software* (27:2), IEEE.

Ali Babar, M., Brown, A. W., and Mistrik, I. 2013. "Agile Software Architecture: Aligning Agile Processes and Software Architectures," *Agile Software Architecture: Aligning Agile Processes and Software Architectures*, Oxford: Newnes.

Angelov, S., Meesters, M., and Galster, M. 2016. "Architects in Scrum: What Challenges Do They Face?," in *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 9839 LNCS), Copenhagen, pp. 229–237.

Babar, M. A. 2009. "An Exploratory Study of Architectural Practices and Challenges in Using Agile Software Development Approaches," in *3rd European Conference on Software Architecture*, pp. 81–90.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., and others. 2001. *Manifesto for Agile Software Development*.

Besson, P., and Rowe, F. 2012. "Strategizing Information Systems-Enabled Organizational Transformation: A Transdisciplinary Review and New Directions," *The Journal of Strategic Information Systems* (21:2), pp. 103–124.

Bharadwaj, A., El Sawy, O. A., Pavlou, P. A., and Venkatraman, N. 2013. "Digital Business Strategy: Toward a next Generation of Insights," *MIS Quarterly*, JSTOR, pp. 471–482.

Canat, M., Català, N. P., Jourkovski, A., Petrov, S., Wellme, M., and Lagerström, R. 2018. "Enterprise Architecture and Agile Development - Friends or Foes?," in *22nd International Enterprise Distributed Object Computing Workshop*.

Dikert, K., Paasivaara, M., and Lassenius, C. 2016. "Challenges and Success Factors for Large-Scale Agile Transformations: A Systematic Literature Review," *Journal of Systems and Software* (119), pp. 87–108.

Drews, P., Schirmer, I., Horlach, B., and Tekaat, C. 2017. "Bimodal Enterprise Architecture Management: The Emergence of a New EAM Function for a BizDevOps-Based Fast IT," in *21st International Enterprise Distributed Object Computing Workshop*, pp. 57–64.

Fowler, M. 2003. "Who Needs an Architect?," *IEEE Software* (20:5), Los Alamitos: IEEE Computer Society Press, pp. 11–13.

Fuchs, C., and Hess, T. 2018. "Becoming Agile in the Digital Transformation: The Process of a Large-Scale Agile Transformation," in *39th International Conference On Information Systems*, San Francisco.

Gerster, D., Dremel, C., and Kelker, P. 2018. "'Agile Meets Non-Agile': Implications of Adopting Agile Practices at Enterprises," in *24th Americas Conference on Information Systems*.

Greefhorst, D., and Proper, E. 2011. *Architecture Principles: The Cornerstones of Enterprise Architecture*, Berlin: Springer.

Hanschke, S., Ernsting, J., and Kuchen, H. 2015. "Integrating Agile Software Development and Enterprise Architecture Management," in *48th Hawaii International Conference on System Sciences*.

Hauder, M., Roth, S., Schulz, C., and Matthes, F. 2014. "Agile Enterprise Architecture Management: An Analysis on the Application of Agile Principles," in *International Symposium on Business Modeling and Software Design*.

Horlach, B., Drechsler, A., Schirmer, I., and Drews, P. 2020. "Everyone's Going to Be an Architect: Design Principles for Architectural Thinking in Agile Organizations," in *Proceedings of the 53rd Hawaii International Conference on System Sciences*.

Kettunen, P., and Laanti, M. 2017. "Future Software Organizations - Agile Goals and Roles," *European Journal of Futures Research* (5:1), p. 16.

Kulak, D., and Li, H. 2017. "The Journey to Enterprise Agility: Systems Thinking and Organizational Legacy," *The Journey to Enterprise Agility: Systems Thinking and Organizational Legacy*, Cham: Springer International Publishing.

Leffingwell, D. 2007. *Scaling Software Agility: Best Practices for Large Enterprises*, Addison-Wesley.

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

Miles, M., Hubermann, M., and Saldana, J. 2014. *Qualitative Data Analysis: A Methods Sourcebook*, Thousand Oaks: Sage Publications.

Rogers, D. L. 2016. *The Digital Transformation Playbook: Rethink Your Business for the Digital Age*, Columbia University Press.

Runeson, P., and Höst, M. 2009. "Guidelines for Conducting and Reporting Case Study Research in Software Engineering," *Empirical Software Engineering* (14:2), Springer, p. 131.

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

Sherehiy, B., Karwowski, W., and Layer, J. 2007. "A Review of Enterprise Agility: Concepts, Frameworks, and Attributes," *International Journal of Industrial Ergonomics* (37:5), Elsevier, pp. 445–460.

Stake, R. 1995. *The Art of Case Study Research*, Thousand Oaks: Sage Publications.

Toth, S. 2015. *Vorgehensmuster Für Softwarearchitektur: Kombinierbare Praktiken in Zeiten von Agile Und Lean*, Munich, Germany: Carl Hanser Verlag.

Uludağ, Ö., Kleehaus, M., Erçelik, S., and Matthes, F. 2019. "Using Social Network Analysis to Investigate the Collaboration between Architects and Agile Teams: A Case Study of a Large-Scale Agile Development Program in a German Consumer Electronics Company," in *Lecture Notes in Business Information Processing* (Vol. 355), Montreal, pp. 137–153.

Uludağ, Ö., Kleehaus, M., Xu, X., and Matthes, F. 2017. "Investigating the Role of Architects in Scaling Agile Frameworks," in *21st International Conference on Enterprise Distributed Object Computing Conference*, pp. 123–132.

Yin, R. 1994. *Case Study Research: Design and Methods (5th Ed.)*, Thousand Oaks: Sage Publications.