

Master's Thesis
for the Attainment of the Degree
Master of Science
at the TUM School of Management
of the Technische Universität München

The Role of Architecture in a Scaled Agile Organization - A Case Study in the Insurance Industry

Examiner: Chair of Software Engineering for Business Information Systems
Prof. Dr. Florian Matthes

Advisor: Ömer Uludağ M.Sc.

Course of Study: Management & Technology M.Sc.

Author: Christina Schimpfle
Lechfeldstraße 10
86931 Prittriching
Matriculation Number 03628567

Date: August 10, 2017

Master's Thesis
for the Attainment of the Degree
Master of Science
at the TUM School of Management
of the Technische Universität München

The Role of Architecture in a Scaled Agile Organization - A Case Study in the Insurance Industry

Examiner: Chair of Software Engineering for Business Information Systems
Prof. Dr. Florian Matthes

Advisor: Ömer Uludağ M.Sc.

Course of Study: Management & Technology M.Sc.

Author: Christina Schimpfle
Lechfeldstraße 10
86931 Prittriching
Matriculation Number 03628567

Date: August 10, 2017

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references.

I am aware that the thesis in digital form can be examined for the use of unauthorized aid and in order to determine whether the thesis as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future theses submitted. Further rights of reproduction and usage, however, are not granted here.

This paper was not previously presented to another examination board and has not been published.

München, den 10. August 2017

Christina Schimpfle

Abstract

Digitization and disruptive innovations change the environment of organizations rapidly. Especially, large organizations face challenges when it comes to developing applications which satisfy the customers' changing requirements at a quick pace. Their original enterprise architecture is characterized by legacy systems with long life and release cycles. The role of architecture is adjusted to this strongly intertwined environment, in which making changes is very complex and happens tediously.

To stay competitive, many enterprises develop applications using agile development methods, such as Scrum or eXtreme Programming. However, Scrum is a method for small, cross-functional teams. As soon as there are several agile teams working on one project, challenges such as cross-team coordination and communication as well as the focus on a common goal become very challenging. Especially, the lack of architectural guidance poses a challenge to the teams and to the success of the entire project. To address these challenges there are agile scaling frameworks, such as Nexus, Large-Scale Scrum (LeSS) and the Scaled Agile Framework (SAFe). In this thesis, these frameworks are analyzed with focus on their approach to architecture. However, their advice on architecture in agile development projects is limited. As two of the frameworks suggest Domain-driven Design (DDD) for architecting in an agile way, this approach is discussed.

So far no approach on how to combine agile scaling frameworks and DDD was defined. The goal of this thesis is to explore how DDD can be established in a scaled agile development project. Therefore, based on a case study in a large insurance company a framework for (enterprise) architecture in agile teams is defined. The framework is largely based on LeSS and incorporates strategic and tactical DDD. The framework is evaluated by applying it in cooperation with three agile teams in the large insurance company. The results are part of the presented pre-study. Additionally, the framework is evaluated in interviews with different stakeholders and roles in the framework.

In the discussion the key findings and limitations are presented. Finally, the research questions relating to the role of architecture in scaled agile organizations are answered and an outlook on future research as well as on practical applications is given.

Contents

Abstract	vii
1 Introduction	1
1.1 Motivation	1
1.2 Research Questions	2
1.3 Approach	3
2 Foundations	7
2.1 Agile Software Development	7
2.1.1 The Agile Manifesto	7
2.1.2 Scrum	9
2.2 Large-Scale Agile Development	16
2.2.1 Benefits and Challenges of Large-Scale Agile Development	16
2.2.2 Template for comparing agile scaling Frameworks	16
2.3 Nexus	17
2.3.1 Framework and Objectives	17
2.3.2 Roles	19
2.3.3 Process	19
2.3.4 Artifacts	20
2.3.5 Nexus and Architecture	21
2.4 Large-Scale Scrum (LeSS)	21
2.4.1 Framework and Objectives	21
2.4.2 Roles	23
2.4.3 Process	24
2.4.4 Artifacts	25
2.4.5 LeSS and Architecture	26
2.5 Scaled Agile Framework (SAFe)	29
2.5.1 Framework and Objectives	29
2.5.2 Roles, Process and Artifacts	29
2.5.3 SAFe Update	34
2.5.4 SAFe and Architecture	34
2.6 Comparison of Nexus, LeSS and SAFe	37
2.6.1 Frameworks and Objectives	37
2.6.2 Roles	39

2.6.3	Process	41
2.6.4	Artifacts	45
2.6.5	Architecture	46
2.7	Domain-driven Design (DDD)	49
2.7.1	Approach and Objectives	49
2.7.2	Domains, Subdomains and Bounded Contexts	52
2.7.3	Establishing an Ubiquitous Language	53
2.7.4	Strategic Design	54
2.7.5	Tactical Design	59
2.8	Related Work	60
2.8.1	Bente et al. (2012)	60
2.8.2	Keller (2017)	60
2.8.3	Dikert et al. (2016)	61
2.8.4	Rost et al. (2015)	61
3	Framework for (Enterprise) Architecture in Agile Teams	63
3.1	Case Study in an international Insurance Enterprise	63
3.1.1	Overview	63
3.1.2	Roles	64
3.1.3	Process	64
3.1.4	Artifacts and Tools	65
3.1.5	Architecture	65
3.1.6	Major challenges	65
3.2	Framework for (Enterprise) Architecture in Agile Teams	68
3.2.1	Agile Scaling, DDD and Architecture	68
3.2.2	Overview	71
3.2.3	Roles	74
3.2.4	Process	78
3.2.5	Artifacts	86
3.2.6	Tools	90
4	Evaluation	99
4.1	Pre-Study	99
4.2	Results of the Interviews	105
4.2.1	General Questions	105
4.2.2	Strategic DDD	107
4.2.3	Tactical DDD	109
4.2.4	Development Process	111
4.2.5	Final Questions	115
5	Discussion	117

6 Conclusion	121
6.1 Summary	121
6.2 Outlook	123
Bibliography	125

1 Introduction

To begin with, the topic of architectural activities in scaled agile organizations is motivated. Based on this, three research questions arise. Additionally, the approach to answer these research questions is described.

1.1 Motivation

The environment of organizations is changing rapidly. While digitization and digital disruption offer new opportunities to companies, they can also pose a significant threat to established companies as their long successful business models are put at risk to be disrupted [69, 87]. Digital disruption describes "changes in the competitive environment resulting from the use of digital technologies by new market entrants or established competitors in ways that undermine the value proposition of a company's product/service portfolio or go-to-market approach" [69]. Today, it is not enough for enterprises to offer the best products or services in the market. Undergoing digital transformation, they must focus on customer experience as their key differentiator [79].

To provide innovative software products and to adapt these to emerging stakeholder needs rapidly, an increasing number of companies uses agile development approaches [18]. Agile development practices promise improvements in software quality as well as productivity and team morale [59]. Hence, the popularity of agile approaches gained significant popularity in the last years in order to respond to dynamic market conditions quickly and effectively [12].

Scrum has become the most wide-spread agile approach for software development in small, self-organizing teams [57, 28, 46, 5]. Large organizations also strive to adopt the benefits of Scrum in their software development [59]. However, as soon as there are several Scrum teams in a complex environment, which are not entirely independent from each other, Scrum cannot be applied without adapting some of its principles, roles and processes [49]. In the last few years, numerous approaches to scale agile methods to large organizations were defined, while still keeping the benefits of small, independent teams [38]. Three of the most popular agile scaling approaches are Nexus, Large-Scale Scrum (LeSS) and the Scaled Agile Framework (SAFe) [80]. These practices focus on agile development with many teams including cross-team coordination and communication [8].

For agile development practitioners, software architecture is part of an initial design process which often cannot be in line with agile principles. In their opinion, defining software architecture upfront, as in waterfall approaches and traditional enterprise architec-

ture management approaches, causes too much work in the beginning, while providing only little value to the customer [12, 83]. In agile projects, architecture is considered to emerge incrementally during the development process through programming [9].

While agile methods have many benefits, especially in large software development projects the lacking definition of the role of architecture - and of architects - can be problematic [12]. However, current literature and practical experience concerning large software projects imply that "some amount of architectural planning and governance is necessary to reliably produce and maintain such systems" [59]. While Nexus and LeSS say that architecture simply evolves in a natural process over time, SAFe acknowledges that emergent design is insufficient in large development projects and therefore also addresses intentional architecture [41]. However, providing overall architectural guidance to agile teams, as implied by intentional architecture, is still a much discussed topic. Agile practitioners fear that architects dictate architectural and technical constraints which limit their freedom and slow down their development speed [59]. Even though the body of research on architecture in agile teams has increased significantly in the last years, the form and subject of architectural guidance for agile teams are still widely unclear [70, 9, 1]. Further, the acceptance of architectural guidance by agile teams is considered to be low [12, 83].

This thesis addresses the role of architecture in agile teams starting with Scrum, agile scaling frameworks and an analysis of their approaches to architecture. SAFe and LeSS address architectural issues using the Domain-driven Design (DDD) approach [41, 20]. Therefore, this method is discussed additionally. The main result of this thesis is a framework for (enterprise) architecture in agile teams, including agile scaling as well as DDD characteristics. The framework describes a lightweight method how enterprise architects can provide architectural and methodological guidance to agile teams. The framework for (enterprise) architecture in agile teams defines roles, a process, artifacts and tools to establish architectural guidance supported by DDD in several agile teams. The approach provides value to the teams, while keeping system architecture sound. The framework is defined and evaluated in a case study in a large insurance enterprise.

1.2 Research Questions

In this section, the research questions (RQ) which are addressed in the thesis are described.

RQ1. What is the role of architecture in scaled agile organizations? The objective of the first research question is to identify practices for scaling Scrum in large organizations and to give an overview of their approach to architecture. In the beginning, a definition of agility according to the Agile Manifesto as well as of Scrum, which is the most common agile development method, is given. Further, it is discussed what makes scaling Scrum in large organizations so challenging. Then three popular agile scaling frameworks - Nexus, Large-Scale Scrum (LeSS) and the Scaled Agile Framework (SAFe) - are explained and compared including their approaches to architecture. The findings are based on a literature

review. The main result answering this research question is a comparison of the practices concerning architecture in Section 2.6.5.

RQ2. How can Domain-Driven Design be adopted in a large organization with several agile development teams? As DDD is recommended by some agile scaling frameworks to be used to support the definition of architecture, an overview of strategic as well tactical DDD is given in the foundations section. As a starting point for the case study in the large insurance company, it is explored how DDD can contribute to agile development involving multiple teams. After a description of the current situation of agile teams and their challenges at the large insurance company, an explanation of the idea on how to integrate DDD into the work of multiple agile development teams and of enterprise architects is presented. Based on the findings, a framework for (enterprise) architecture in agile teams is defined. The result is not a framework for enterprise architecture, such as TOGAF or Zachman. Instead the framework describes how enterprise architects can support agile teams with their practices, new methods and knowledge based on DDD.

RQ3. Which roles, processes, artifacts and tools are required for scaled Domain-Driven Design? In the literature review, the main roles, processes and artifacts of the agile scaling frameworks are identified and compared to build the basis for answering this research question. This as well as the current situation of agile teams at the large insurance enterprise are the basis for defining a framework for (enterprise) architecture in agile teams including the required, roles, processes, artifacts and tools. The focus is put on establishing strategic and tactical DDD. The roles of agile team members, project managers and enterprise architects in the process are elaborated. Additionally, tools to support the application of the framework are described and applied. The part of the framework concerning strategic DDD is evaluated in a pre-study. The pre-study is conducted to make the first steps taken to establish strategic DDD in a part of the company with agile teams apparent. Finally, the framework is evaluated by conducting interviews with the key roles in the agile teams, architects as well as decision makers in the large insurance enterprise.

1.3 Approach

The approaches used in this thesis comprise a literature review and a case study. The aim of the literature review in this work is to investigate the role of architecture in scaled agile organizations by examining its role in three selected agile scaling frameworks. The practices and the role of architecture are investigated by a literature review consisting of five phases [85]. First, the scope and suitable research questions are defined. The approach to selecting the practices that are analyzed in detail is to consider size and scope of the practices as well as their popularity. Further, it is important to examine practices with different approaches to architecture to find a suitable one for the company in the case study and for

the later defined framework for (enterprise) architecture in agile teams. Based on these criteria Nexus, Large-Scale Scrum and the Scaled Agile Framework are chosen. The results of the literature review contribute to answering all three research questions. However, only the first research question is purely addressed by the literature review.

In the next step, suitable definitions and summaries of the practices are examined to gain a general understanding. In this phase, mostly books and whitepapers about the practices are included.

It aids with further defining relevant search terms which are used in the third phase. The main search terms are Nexus, Large-Scale Scrum, Scaled Agile Framework, scaling agile, Scrum and agile architecture. In this step, the relevance of DDD for architecture in agile teams is recognized. Therefore, this topic is explored additionally based on relevant literature. The databases EBSCOhost, Scencedirect, IEEE Explore, SpringerLink, Scopus, Safari Books Online (ProQuest) and Google Scholar are used to find articles and books about the relevant topics. Additionally, relevant sources of the analyzed papers are searched.

In the fourth phase, the results are structured and evaluated. The three practices are compared systematically using the dimensions objectives, roles, processes, artifacts and architecture. The last phase builds the basis for roles, process, artifacts and architecture to be addressed in the case study [85].

Based on the gained insights in the literature review a case study is conducted with the result of a framework for (enterprise) architecture in agile teams. The framework is defined based on the literature, but adapted based on the insights the case study provides.

A case study is an approach with focus on understanding the dynamics in certain situations. In a case study, different methods of data collection can be utilized, such as interviews and observations [29]. The case study helps to evaluate if and how such a framework is helpful in the chosen setting at the large insurance company. According to Yin, a case study has six phases [90].

In the planning phase, it is evaluated if the method of the case study is a suitable approach. As a part of the design phase, it is decided if one or multiple cases should be considered. Here, a single case is examined. In the following, the case to be evaluated is chosen. In this thesis, three of the agile teams at the large insurance company participate in the case study, in which certain components of the framework are tested in cooperation with selected members of these teams. This leads to the data collection which takes place in form of a pre-study and interviews. The pre-study addresses only a certain part of the framework (strategic DDD) and is evaluated statistically. Then the results are discussed with relevant roles in the company to ensure validity and reliability of the approach. The structured interviews are conducted with nine persons in different roles to evaluate all components of the framework. The next step is to analyze the collected data. Finally, the results are visualized and presented to the affected and relevant roles within the company [90]. Figure 1.1 gives a comprehensible overview of the research approach.

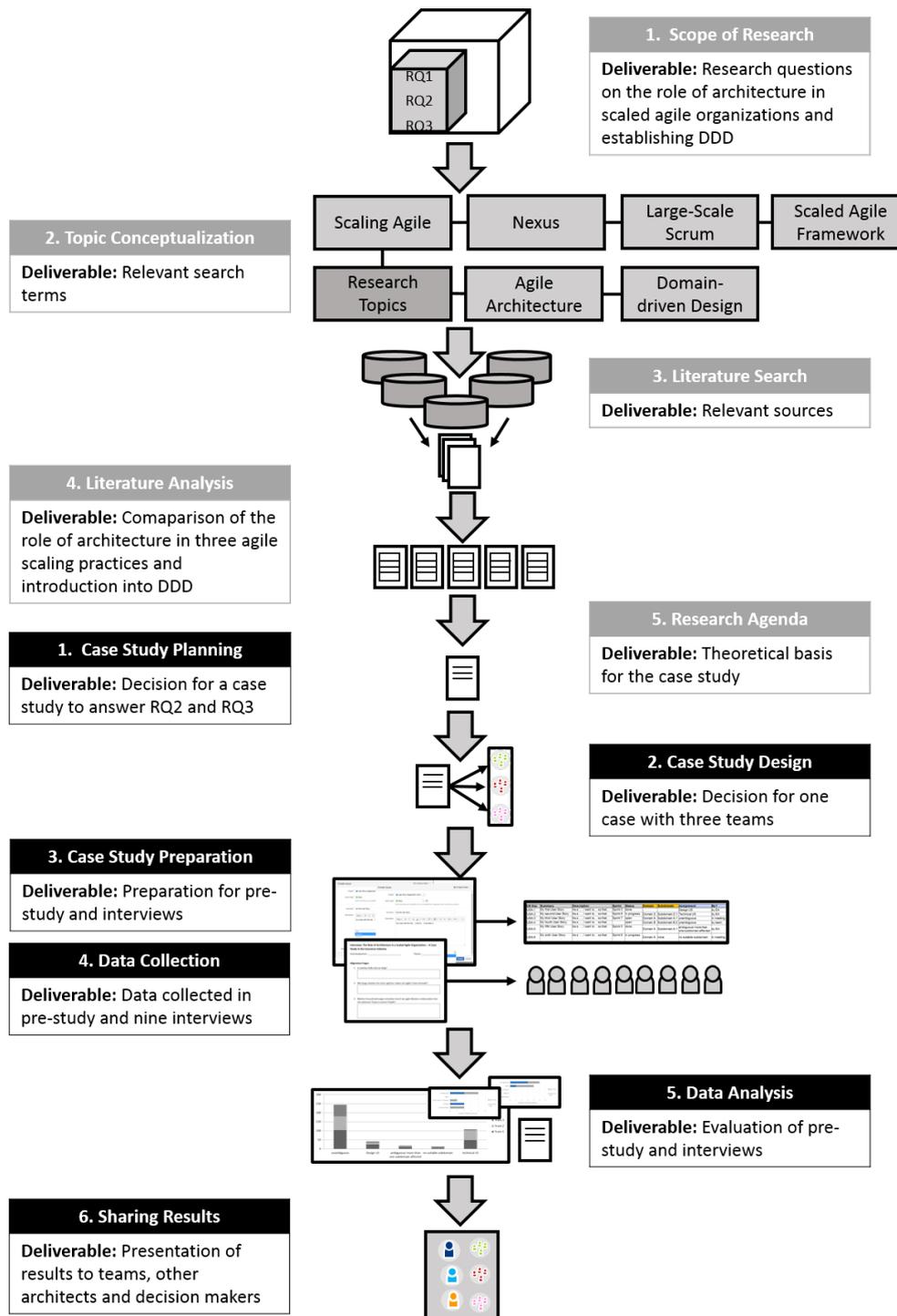


Figure 1.1: Literature review and case study

2 Foundations

This chapter gives a theoretical overview of the most important concepts which are addressed and utilized in this thesis. First, the Agile Manifesto is presented to describe what agility is. Scrum, as one of the most popular agile development methods, is explained. After describing the major challenges of applying Scrum methodology in large organizations three approaches for scaling agility are presented: Nexus, Large-Scale Scrum and the Scaled Agile Framework. The explanations of these practices include their main objectives, roles, processes and artifacts. Central for answering the first research question, the role of architecture in these practices is described. Finally, the three agile scaling frameworks are compared concerning roles, processes, artifacts and architecture. Finally, the main concepts of DDD are introduced.

2.1 Agile Software Development

The following section describes agility in Software development and Scrum as one of the widest spread agile development approaches. Additionally, a template for presenting and comparing the following agile scaling frameworks is shown.

2.1.1 The Agile Manifesto

In 2001, experts and practitioners of agile development methodologies defined the Agile Manifesto including four core values and twelve principles to produce excellent software in an agile way [57]. The core values are:

- "Individuals and interactions over processes and tools" (1)
- "working software over comprehensive documentation" (2)
- "customer collaboration over contract negotiation" (3) and
- "responding to change over following a plan" (4) [3].

The four core values include that individuals and their interactions are more important than processes and tools (1). Software is build by individuals who communicate with each other, should be motivated and work self-organized.

Progress is measured by working software which is more crucial than having a comprehensive documentation (2). The goal is to provide business value to the customer by frequently providing software. Documentation should be only done if it adds value.

Collaborating with the customer is more essential than contract negotiation (3). To reach the customer's vision the development team has to collaborate closely and continuously with the customer.

Being able to respond to changes throughout the development process is more important than following a predefined plan (4). If the customer's requirements change, the team has to be able to adapt [35].

Furthermore, the Agile Manifesto comprises twelve core principles as presented in Table 2.1.

	Core Principles
P1	"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."
P2	"Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage."
P3	"Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale."
P4	"Business people and developers must work together daily throughout the project."
P5	"Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done."
P6	"The most efficient and effective method of conveying information to and within a development team is face-to-face conversation."
P7	"Working software is the primary measure of progress."
P8	"Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely."
P9	"Continuous attention to technical excellence and good design enhances agility."
P10	"Simplicity - the art of maximizing the amount of work not done - is essential."
P11	"The best architectures, requirements, and designs emerge from self-organizing teams."
P12	"At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."

Table 2.1: Core principles in the Agile Manifesto [3]

The first principle defines how to provide value to the customer by delivering software early in the project and continuously as the highest priority in agile software development (P1). Second, agile teams need to be aware that the customer's requirements change over time and need to be able to implement those changes even late in the development process to support the customer's competitive advantage (P2). Third, working software should be delivered as frequently as reasonable and possible, e.g. every couple of weeks or month (P3). Collaboration between business experts and developers is essential and is practiced

daily (P4). Building an agile team requires motivated people. The individuals need to be provided an appropriate environment and support to keep them motivated. Putting trust in the team to succeed is very important (P5). Face-to-face communication is the most effective and efficient way to exchange information and to gain deep understanding (P6). The progress of a project is measured by the amount and quality of working software, not by models or documentation (P7). Agile processes foster sustainable delivery. The agile team needs to be able to keep its pace and productivity constant over the complete length of the development project. Working longer hours does not mean higher productivity (P8). The agile team needs to pay attention continuously to good design which is enhanced throughout the development process (P9). Tenth, keeping the software simple is crucial to be able to add functionality throughout the iterations. Implement only what is necessary to many stakeholders and avoid unnecessary complicated solutions (P10). The teams should be self organizing which is supposed to provide the best architecture and design (P11). Finally, the agile team needs to reflect on how to adjust its behaviour to become more effective (P12) [3, 35, 57].

The Agile Manifesto describes a set of values and principles which are helpful for development teams, but it is important to be aware that it does not provide a common methodology for all software development projects. While project teams can benefit from the Manifesto, each project has to define the most suitable methodology to achieve its goals [35].

2.1.2 Scrum

This section gives an overview of Scrum including its main objectives, as well as roles, the process, artifacts, tools and architecture in the agile development process.

Objectives Based on the core values and principles of the Agile Manifesto many different agile methodologies exist, such as Scrum, Extreme Programming (XP) and Crystal [12]. Scrum has become one of the most popular over time [57, 28, 46, 5]. Based on the assumption that small independent teams work more effective, Scrum is a iterative product development process for small teams. Focus on a common goal for the whole team is essential to Scrum. The team works together closely, knowing the priorities and each of the members having a well-defined role knowing his/her own tasks in each product increment [68]. The objective of Scrum is to develop products, mostly software, of high-quality while maintaining flexibility to be able to react to changes in the environment and of the requirements during the whole development process. Initially, in Scrum only the context and a broad definition of the deliverable are determined while being aware that this definition is incomplete and that the environment as well as the deliverable evolve throughout the development process [72].

Scrum as an agile methodology is consistent with the four core values of the Agile Manifesto. Especially, the focus on team responsibility is essential ("Individuals and interactions over processes and tools" [3]). As soon as a Scrum team knows its business goal, it has

to figure out how to reach the goal, do the work, identify possible challenges, resolve all challenges within its scope and collaborate with other business units to resolve those difficulties which are outside the team's control [7]. After every sprint the team is to provide a working product increment which does not need to include all functionality necessary to ship the product, but the already implemented functionality needs to be of shippable quality ("Working software over comprehensive documentation" [3]) [7]. Additionally, promoting and facilitating collaboration is critical for successful Scrum ("Customer collaboration over contract negotiation" [3]). While the team members collaborate with each other to detect the most suitable way to build a software, they also have to collaborate with other stakeholders to achieve a valuable software deliverable which provides business value and complies with the customer's vision [7]. Every Scrum team has to plan the current sprint. Additionally, some Scrum teams define long term plans to support the business and the team itself when making decisions. In Scrum the plan itself is not a central concept. The thought process during planning including gathering new ideas is far more important. Furthermore, the team is not supposed to stick blindly to the plan if changes occur ("Responding to change over following a plan" [3]). Not only the plans, but also the product backlog is continuously updated to react to changes or to incorporate newly discovered information. Responding to changes increases the chances of developing a successful product with maximum business value which fits the current environment and the customer's vision [7].

The central Scrum values which are mostly relate to collaboration in the team are focus, courage, openness, commitment and respect. In Scrum it is essential to focus only on a few tasks at the same time to fulfill them quickly and in the best possible way. Courage means the team should stick together closely and support each other to be able to solve even the most challenging problems. Being open means that the team members openly talk about the challenges they face and discuss how to address them. This not only relates to the tasks to be worked on for the product increment but also how the team works together and how every team member is feeling. Commitment to success is essential because the team in Scrum should be able to work very independently from other organizational units and therefore is the only one that can reach its own goals. Respect between team members includes treating each other respectfully as well as supporting each other and to take shared responsibility for failures and successes [7].

Scrum is the basic development methodology in practices which have the objective to make large enterprises more agile that are presented in the following sections. Furthermore, the agile teams in the case study use Scrum with some adaptations as their methodology. Figure 2.1 shows a basic Scrum process including the most essential roles and artifacts.

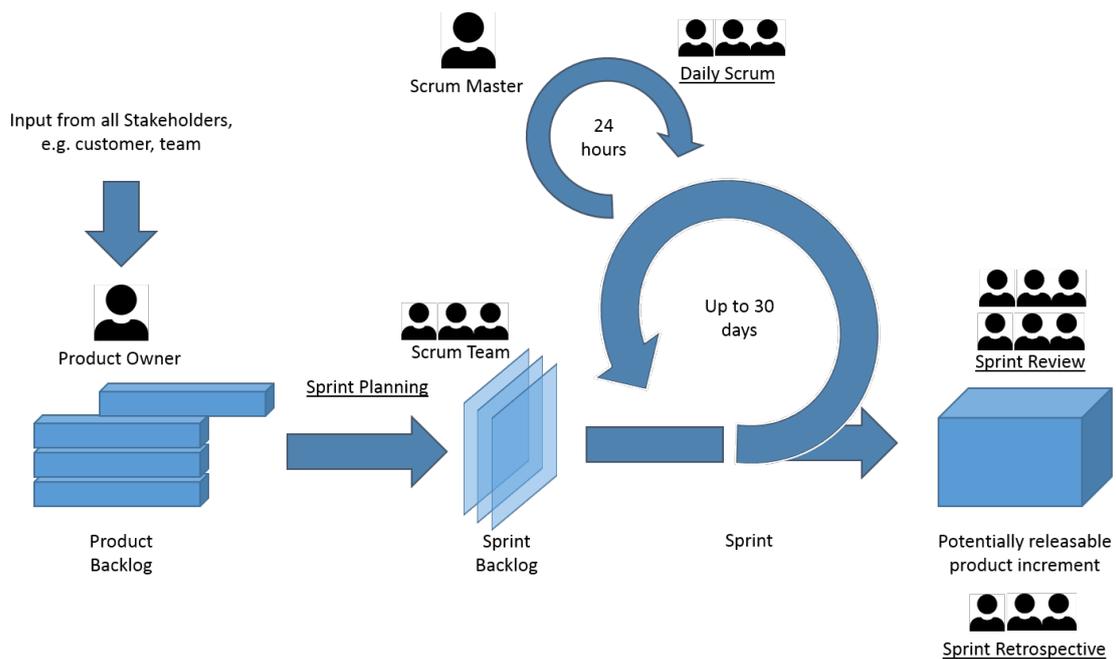


Figure 2.1: Scrum [57]

Roles A Scrum team consists of a product owner, the Scrum master and ideally three to nine developers. A bigger team causes higher complexity and need for much more coordination, while smaller teams may have a skill set which is not diversified enough to solve all problems the team encounters. The teams are cross-functional to include all competencies to be able to achieve their goals independently from others outside the team. Additionally, Scrum teams are self-organizing which means that the team members themselves determine how to achieve their overall goal. This characteristics foster the optimization of productivity, flexibility and creativity [6].

The product owner makes sure that the business value of the created product is maximized and is responsible for the product backlog. He/She represents the customer to the development team. Everyone who has a requirement for the product or wants to change a requirement's priority has to address the product owner. The product owner clearly describes all requirements as items of the product backlog and orders them to achieve the team's goals and to optimize the work of the development team. The development team only works on requirements specified by the product owner [6].

The development team delivers the increment at the end of each sprint. It is best practice in Scrum to organize a team around features. Feature teams are long-lived, stable over time, cross-functional, cross-component, develop end-to-end features and consist of co-located members in the best case [53]. Team members organize their work themselves. The team decides how to implement the requirements from the product backlog and delivers the

functionalities. Even if the developers have different specialized competencies and areas of focus, there are no sub-teams within a team and the whole team is accountable for all parts of a product increment [6].

The Scrum master makes sure that the development team applies Scrum theory, rules and practices to ensure the success of the project. The Scrum master is not a classical leader to the team, but provides mostly methodological guidance to both the development team and the product owner. The Scrum master supports the product owner to create a valuable product backlog and he/she supports the development team to implement the specifications from the product backlog. He/she makes sure all Scrum meetings take place according to Scrum rules [73].

Process In Scrum all work is done in short development phases called sprints which last from one up to four weeks [68]. The duration of the sprints does not change throughout the development process and after the end of a sprint the next one starts immediately. The result of each sprint is a product increment which potentially could be released. Each sprint includes the events sprint planning, daily Scrums, the actual development, a sprint review and retrospective.

In the sprint planning the team determines collaboratively what will be included in the product increment at the end of the sprint and how the determined goal will be achieved. As all events in Scrum the sprint planning is time-boxed to a maximum of eight hours for a 30 day sprint with the preference to keep it shorter [6]. After committing the scope in the sprint planning adding further functionalities is not possible except by the development team [57].

The daily Scrum is used to inspect the progress toward the defined sprint goal and to synchronize work until the next daily Scrum on the following day. Every member of the development team explains what he/she has done since the last daily Scrum and until the next daily Scrum as well as potential obstacles preventing the team from reaching the sprint goal. The daily Scrum helps the team to work together to remove those obstacles, improves communication and knowledge within the team and allows making decisions quickly without additional meetings taking place. The daily Scrum takes place at the same time everyday and is time-boxed to 15 minutes [6].

At the end of each sprint the Review takes place to inspect the product increment to be able to incorporate feedback and to adapt the product backlog. The new product increment is demonstrated, it is discussed what has (not) been done and what to do next. This event is attended by the development team, the product owner and key stakeholders and is time-boxed to four hours for a 30 day sprint [6].

A further Scrum event can be the backlog Refinement in which the product owner and the development team adapt the product backlog by adding new requirements, describing requirements more in detail and by ordering the requirements according to their priorities. This can also be done in the sprint review [6].

The sprint retrospective after the end of each sprint and before the sprint planning of the

next sprint is used to inspect what went well and what not regarding the team, product, process and tools. Potential improvements are identified and the team defines how to implement the improvements. The definition of "done" is adapted if necessary. The retrospective is time-boxed to three hours for a 30 day sprint and is attended by the development team and the Scrum master [6].

Artifacts The artifacts created in Scrum are mostly important to reach transparency and to detect opportunities for further inspection and improvement. The artifacts in Scrum are the product backlog, the sprint backlog, a definition of "done" and the product increment itself [6].

The product backlog includes a list of all functionality to be provided by the product which is developed by a team. The product owner takes responsibility for its availability, content and priorities of the single requirements. At the start of a project the product backlog is defined including all requirements which are clear at that time. However, the product backlog is a dynamic artifact which is adapted over time. Requirements may be changed, added or deleted if they are not considered necessary anymore [73]. The items which are placed higher in the product backlog need to be described more in detail than the ones which have lower priorities. The items which might be part of the next sprint have to be defined in detail and need to be doable in the timeframe of one sprint. The product owner can track progress by analyzing what is already done and how many requirements remain in the product backlog. The product backlog exists not only during the time when the product is developed but throughout its whole lifetime [6].

The sprint backlog includes the work to be implemented in the upcoming sprint to reach the sprint goal by providing a potentially releasable product increment. In the sprint planning the team decides which requirements are part of the next sprint. The team estimates how long it takes to finish each of the requirements. The tasks to be executed should be divided in pieces of work that take between 4 and 16 hours to be implemented. Tasks which take longer need to be divided in several smaller tasks [73]. This happens at best before the sprint planning. Based on the estimated duration of the tasks and the working days of each team member the team determines how many and which requirements can be turned into software functionality in the next sprint. During the sprint the team keeps the sprint backlog updated by assigning requirements to team members and by labelling requirements which are already implemented and need to be tested or are entirely done. To meet the sprint goal all requirements in the sprint backlog need to be implemented at the end of the sprint.

Another artifact is the Product Increment itself which represents the sum of all requirements implemented in the previous sprints. At the end of each sprint the product increment is supposed to be "done" meeting the team's definition of "Done". The definition of "done" is determined and understood by the team members. It describes when work is complete and can additionally include criteria for high quality. The definition of "done" evolves over time to include a more detailed description [6].

Not being artifacts defined by the Scrum methodology itself, user stories are nevertheless often used as a form to describe requirements in the Product and sprint backlog. A user story comprises a short and simple written description, communication about the story to refine it and acceptance criteria to define when a user story is complete. The acceptance criteria can be used for testing. A user story is formulated from the viewpoint of the user and his/her role in the developed Software and includes what the user wants to do and why this provides value [62]. An example of good user story could be "As an user, I want log in with my user name and password to get access to the system". Additionally, acceptance criteria could say "Access is granted when using a correct user name and password combination, otherwise not".

Further, user stories should follow the INVEST-criteria which state user stories should be Independent, Negotiated, Valuable, Estimatable, Small and Testable. These criteria are explained in Table 2.2.

	INVEST
Independent	... means that the user stories should be as independent from each other as possible, so that the order in which they are implemented does not matter.
Negotiated	... says that their content is based in close collaboration and negotiation between the development teams and key stakeholders.
Valuable	... means that a user story needs to provide value to the users and that this value must be described clearly in the story.
Estimatable	... user stories allow the development team to estimate how long it will take to implement the user story by assigning story points.
Small	... means that user stories should not be too big or too small. Those stories either need to be split or aggregated into one story.
Testable	... means that the teams needs to be able to determine if it was completed correctly.

Table 2.2: INVEST-criteria [19, 62]

Furthermore, user stories should be atomic, minimal, problem-oriented, unambiguous, well-formed and conceptually sound [62]. User stories are often written on sticky notes or index cards, but there exist also some tools to manage user stories digitally [19].

Tools The Scrum Guide and other central sources concerning Scrum methodology do not describe tools that support the application of Scrum [6, 73, 74]. However, teams which apply Scrum can be supported by different tools. The functionality of Scrum tools includes the possibility to manage the product backlog being able to describe and prioritize backlog items. A tool should also comprise a clear visualization of the items which are part of

the sprint backlog and therefore the current sprint. Further, tools need to provide the functionality to assign items to team members and a status concept to document which items are already finished. Additionally, visualization tools, such as burndown charts which represent the work which is left to do (vertical axis) along the timeline of the project (horizontal axis) are helpful [86]. Next to the described functionality, usability and the configuration possibilities of the tool are important [81]. As in Scrum literature, not much information could be found about tooling for the scaling practices. Therefore, the topic of tooling will not be described in detail in this thesis.

Scrum and Architecture Most agile methods, such as Scrum, do not provide particular guidance concerning the architecture of the developed solution [57]. The Scrum methodology includes the assumption that architecture evolves naturally during the iteration cycles along with the implementation of further functionality and continuous re-factoring [59]. None of the roles in Scrum is described to have detailed architectural knowledge or responsibility.

As Scrum relies on the Agile Manifesto, the principle "the best architectures, requirements, and design emerge from self-organizing teams" is the central assumption for architecture in Scrum [3, 56]. Even if during an initial planning phase a possible target architecture is defined, it will change over time [68]. Practitioners of agile approaches often see the upfront design of software architecture as too costly and think it provides only little value to their customers. However, more and more literature on how to integrate agile and architectural practices is published [12]. How much architectural action is required in Scrum teams depends on many factors, such as team distribution, number of agile teams, enterprise size, size of the application and its dependencies on other systems and projects. Especially, large complex projects have a need for high architectural effort [80, 1].

Further, the role of the architect in a Scrum team can be defined differently. The "Internal/Team architect" can be a member of the development team who is an architecture expert sharing his knowledge with the rest of the team. The architect role can also be taken on by the whole team if there is no dedicated expert in the team [9]. Then the architecture emerges as a result of the collaboration of all team members [37]. However, the architecture can also be defined by an external architect who is not a member of the Scrum team. In this setup, the architect can be a partner to many agile teams and is supported by other architects, e.g. in architectural boards [9]. The external architect provides high-level architectural guidance to the teams and defines what high quality architecture means and how the teams can adopt it [37]. Further, there can be two architects, an internal and an external architect. Then the external architect focuses on the big picture, its requirements and the relating decisions as well as being an external coordinator (referred to as "Architectus Reloadus" by Martin Fowler [33]) [9]. His main goal is to ensure conceptual integrity. He makes all the important decisions that need to be made in the beginning of a project [33]. On the other hand, the internal architect (referred to as "Architectus Oryzus" by Martin Fowler [33]) is always aware of what is going on in the team and the code to be able to

solve potential problems before they occur. This requires very intense collaboration with the development team. This kind of architect is also a mentor to the development team so that the team itself is trained in architecture to be able to take on more complex issues themselves and to support the architect to make him not the only decision maker being an architectural bottleneck [33].

The role of the architecture becomes especially critical with many Scrum teams in large organizations. Agile development practices for large scale organizations and their approach to architecture are discussed in the next sections.

2.2 Large-Scale Agile Development

The principles and approaches as defined by the Agile Manifesto and Scrum were created for small teams with small problems to solve and do not scale easily to large organizations with many Scrum teams or large teams with complex business problems. Agile methods became increasingly popular during the last years and large organizations want to profit from the benefits of agile methods, too [38].

2.2.1 Benefits and Challenges of Large-Scale Agile Development

The benefits of agile methodologies comprise a higher velocity in development projects and a resulting shorter time to market for new products. Further, agile approaches contribute to high quality software [79]. Scrum cannot be applied in large organizations without adapting some of its principles, roles and processes [49]. There are numerous approaches to scale agile methods to large organizations while still keeping the benefits of small, independent teams [38]. Three of the most popular ones are Nexus, the Scaled Agile Framework (SAFe) and Large-Scale Scrum (LeSS).

However, large organizations face various challenges when scaling agility, such as inter-team coordination, many dependencies on other projects or applications, lack of clearly defined requirements and agile culture [65]. Especially, a lacking definition of the architecture - on a team level as well as overall - causes problems in large organizations when adopting agile methods. Scrum itself does not provide guidance on architecture but assumes that it emerges on team level naturally with each iteration and continuous re-factoring. This can be problematic as soon as large-scale systems or many different applications are built by many teams which is the case in large organizations. Then at least some governance and architectural planning is required to develop reliable, expandable and scalable systems [59]. In the following the practices Nexus, SAFe and LeSS and their approaches to architecture in large-scale agile organizations are presented.

2.2.2 Template for comparing agile scaling Frameworks

In the following section, three frameworks for scaling Scrum to large software development projects are presented and compared. The description of the frameworks follows a

template including the same major points as the description of Scrum in Section 2.1.2. The template follows the structure used in the Scrum and Nexus guide plus an architecture Section [6, 76]. The template with all its criteria is shown in Table 2.3.

Objectives	Roles	Process	Artifacts	Architecture
Number of Scrum Teams	Product Owner	Sprint	Product Backlog	Emergent Design vs. Intentional Architecture
Coverage of all "levels"	Scrum Master	Product Backlog Refinement	Sprint Backlog	Role of IT Architects
Inter- Team Coordination	Teams	Sprint Planning		Role of Enterprise Architects
Main Objective	Additional Roles	Daily Scrum	Sprint Goal	Architecture in the Development Process
Criticisms		Sprint Review	Product Increment	Events for Designing/ Architecting
Key strengths		Sprint Retrospective	Definition of "Done"	Tooling/ Modeling
		Additional Events	Additional Artifacts	Recommended Practices

Table 2.3: Template for comparing agile scaling Frameworks

Therefore, first a general overview of each of the frameworks themselves and their objectives is given. Then the role of the product owner, Scrum master and development teams in the respective framework are explained. If a framework includes additional roles, these are presented as well. Subsequently, the process and its most essential elements and events, such as sprints, backlog refinement, sprint planning, daily Scrum, sprint review, sprint retrospective as well as additional events are presented. Further, the most important artifacts in the framework are depicted, such as product backlog, sprint backlog, the product increment itself, a definition of "done", sprint goals and other useful artifacts defined by the practice. Lastly, an overview of the role of architecture in the respective framework is given.

The following sections about Nexus (Section 2.3) and LeSS (Section 2.4) are structured according to the presented template into the subsections framework and objectives, roles, process, artifacts and the respective framework and architecture. For the SAFe it seemed to be more comprehensive to describe the roles, process and artifacts in one section (2.5.2) divided by the different levels. The defined dimensions are also used to compare the frameworks in Section 2.6. The comparison of the role of architecture takes place in Section 3.2.1.

2.3 Nexus

The following section gives an overview of Nexus including its central objectives, roles, the process, artifacts and architecture in Nexus.

2.3.1 Framework and Objectives

Nexus means "Unit of development" [76]. The Nexus Framework is a framework for large-scale product or software development consisting of roles, a process with focus on events and artifacts. Nexus is largely based on Scrum and was developed by Ken Schwaber - the

developer of Scrum - and Scrum.org in 2015 [76]. It supports large-scale software development in which three to nine Scrum teams work on a single product increment using a single product backlog [38]. The basic building block of Nexus is Scrum. Therefore, Nexus is consistent with the Scrum methodology and the idea of agility in the Agile Manifesto concerning its basic values, such as lean, working in self-organizing teams, empowering and trusting team members and delivering a working product increment after every sprint [78].

Nexus is an exoskeleton resting on top of three to nine Scrum teams that are dedicated to the development of one integrated "done" product increment supporting them to deal with dependencies and interoperation between the teams. The dependencies between the teams relate to requirements, software and test artifacts as well as domain knowledge. Requirements may affect each other or overlap. This must be taken into account when ordering and selecting requirements for a sprint as well as for implementation and testing. The team members have know-how about different parts of the business and different computer systems. This domain knowledge needs to be considered for the team composition to reduce dependencies between teams and optimize productivity [76].

Nexus is only applicable for three to nine teams. If more teams are involved, Nexus+ is a further extended framework for more than nine teams. In Nexus+ several Nexus units exist, that work as independently from each other as possible. The rules defined by Nexus apply. Unfortunately, there are no further sources including a more detailed description of Nexus+. Therefore, the focus lies on the simpler Nexus framework which is shown in Figure 2.2 [75, 48].

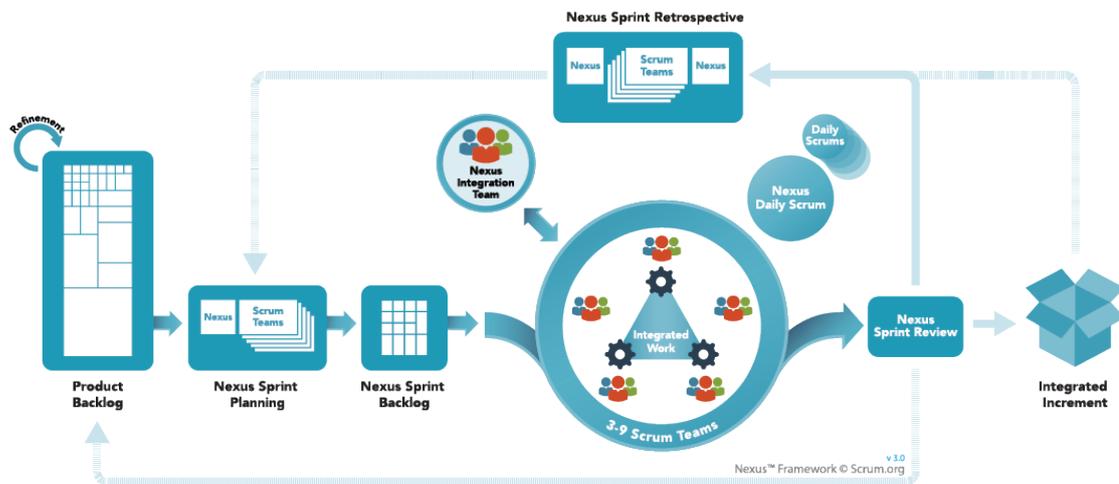


Figure 2.2: Overview of Nexus including roles, process and artifacts [76]

Schwaber explains that they defined more than 30 or even 40 practices for Nexus to op-

erate as aimed, calling those practices Scaling Professional Scrum [75, 60]. Those practices are concerned with development, management, organization and leadership. However, there is no source which names or explains those practices and it seems that the practices are only explained to those who take part in the Scaled Professional Scrum workshops and courses offered by Scrum.org [75]. The practices represent what is needed for scaling software development and the tools are used to automate them [60]. However, as no access to the practices could be gained, only the basics of Nexus as defined by the Nexus Guide are described.

2.3.2 Roles

The roles in Nexus for three to nine Scrum teams are the same as in one-team Scrum plus a Nexus integration team. There are development team members and a Scrum master in every development team and in the Nexus integration team, but a single product owner for all teams. The Nexus integration team supervises the application of Nexus, coaches and coordinates the teams and takes responsibility for all integration issues. This team resolves all non-technical as well as technical constraints which inhibit the Nexus to deliver a "done" product increment. The product owner, one Scrum master and some further appropriate integration team members represent the Nexus integration team. The integration team members can be part of the Scrum teams and can change depending on the current priorities in the Nexus. If this is the case, their work for the Nexus integration team takes precedence over their work in the individual Scrum teams as all teams are affected by the work of the Nexus integration team. The integration team members have knowledge about infrastructural and architectural standards of the organization, detect dependencies between the teams and coach the Scrum teams about necessary tools and practices. The Scrum master in the Nexus integration team is responsible that Nexus is enacted and can be at the same time Scrum master of one or more of the development teams. Next to that there are further Scrum masters who are responsible for the development teams. While the Scrum master in the integration teams takes on overall responsibilities, such as overall integration and use of Nexus practices, the other Scrum masters are responsible that Nexus is enacted in their teams. The product owner is responsible for the single product backlog. To sum up, the roles and their responsibilities in Nexus are quite similar to Scrum plus the additional Nexus integration team [76].

2.3.3 Process

In Scrum as well as in Nexus the work is done in sprints. For Nexus, some of the events of Scrum are conducted as known, extended or replaced by a Nexus event to serve not only the individual teams but the overall success of the common product. In the Nexus sprint planning appropriate representatives of all Scrum teams and the product owner validate the current status of the product backlog and make adjustments if necessary. After that all team members participate to determine the Nexus sprint goal and to coordinate which re-

quirements each of the Scrum teams will work on in the upcoming sprint. Afterwards each team conducts its own separate sprint planning. If new dependencies are detected, they should be visualized and the sequence of the work might need to be adjusted. However, if the product backlog is refined properly, all dependencies should be clear beforehand. All work items which are part of the upcoming sprint for any team become part of the Nexus sprint backlog. In the Nexus daily Scrum appropriate representatives of each Scrum team participate to detect and discuss integration issues as well as dependencies. They focus on the impact each of the teams has on the integrated increment. Following the Nexus daily Scrum, each team holds its own daily Scrum as defined by Scrum. Tasks which arise in the Nexus daily Scrum will be taken back to the intra-team daily Scrum to be considered in the planning for the day. The Nexus sprint review replaces individual sprint reviews by every team because it is used to show all new features in the new integrated increment which can be demonstrated to the stakeholders to be able to incorporate feedback. Finally, a Nexus sprint retrospective is conducted which consists of three parts. Firstly, appropriate representatives from different Scrum teams identify issues which had impact on more than one Scrum team, making those issues transparent and detecting possibilities to deal with them. Secondly, each development team holds its own retrospective possibly using input from the first part of the retrospective. They formulate actions which should be taken to improve in the next sprint. Thirdly, representatives from all development teams visualize and discuss the identified actions. Refining the product backlog is essential to Nexus as the backlog items need to be independent from each other to be worked on by various teams without interruptions. How often and how long the refinement meetings take place depends on the dependencies and the complexity. Backlog items are refined and decomposed from large and very vague requirements to actionable backlog items. It also helps to determine which team is suitable to work on which backlog item. The first half of the Refinement meeting should be used to decompose and detail items while the second half should be used to identify and visualize interdependencies [76].

2.3.4 Artifacts

As in Scrum, the Nexus product backlog is the central artifact including all requirements which need to be implemented by all teams. In Nexus, there is a single backlog for all Scrum Teams which is maintained by the product owner. Additionally, to the backlog in Scrum, the Nexus product backlog serves to detect dependencies and to determine which team will work on which backlog item. Equivalently to Scrum, there is the Nexus sprint backlog which is composed of all items which are worked in a sprint. Therefore, it is an composite of the sprint backlogs of all teams in the Nexus. It is updated regularly and conduces to make dependencies and the flow of work during the sprint transparent. Another artifact is the Nexus sprint goal, which is the overall goal of a sprint for all teams to be reached in collaboration. While each Scrum team has its own sprint goals, these goals are aggregated to the Nexus sprint goal, as the overall objective to be fulfilled. The integrated increment is the integrated product developed by all teams. As is Scrum, a

definition of “done”, which is known by all team members, exists. Summing up, the Nexus artifacts are very similar to the artifacts defined by the Scrum methodology [76].

2.3.5 Nexus and Architecture

Equivalently to Scrum, for Nexus the role of architecture and the role of architects themselves are not defined in the Nexus guide. In contrast to Scrum, Nexus is a quite new framework about which only a small amount of literature was published [80]. Therefore, not much about architectural implications of applying Nexus can be said. Some of the above mentioned practices might relate to high-level architecture, but no information was found about those practices [37]. Therefore, the assumption is that Nexus places a heavy emphasis on emergent architecture. As no concrete practices and implications are given it is assumed that whatever practices help the teams to keep the architecture simple and make the teams successful can be chosen. Giving no guidance for architecture or an idea on how to structure the teams can become problematic if Nexus is applied in large development projects. However, the Nexus integration team which is responsible for the overall integration into a product increment should provide some architectural guidance and align the teams [76].

2.4 Large-Scale Scrum (LeSS)

The following section gives an overview of Large-Scale Scrum with its central objectives, roles, the process, artifacts and the role of architecture for agile scaling.

2.4.1 Framework and Objectives

The first book on the Large-Scale Scrum Framework, short LeSS, was published in 2008 by Craig Larman and Bas Vodde though the authors already had teamed up from 2005 on to apply Scrum in large organizations [51, 54]. From this starting point two forms of LeSS evolved: LeSS for scaling up to eight teams and LeSS Huge for more than eight teams up to a few thousand people that all are working on a single product. LeSS does not intend to be improved Scrum or Scrum applied on the team level with additional layers on top. The focus of LeSS is to apply Scrum principles, processes and elements in a large-scale context. To do so, a deep understanding of one-team Scrum is necessary. LeSS is scaled Scrum adopting many of the principles and practices of Scrum. One of the most crucial objectives of LeSS is to support the different teams to focus on the whole common product instead of just making their part of the product work [21]. The authors of LeSS say that it is not a complete framework, but that it leaves space for each team and organization to learn from its own experiences. The LeSS Framework is defined by a set of rules concerning the organizational structure, the product and sprints which are minimalistic and do not give definitive answers on how to apply LeSS in a certain context.

Further, the framework includes experiments which are practical applications of different forms of LeSS in real organizations and learnings from those experiments. Additionally, LeSS provides guides which are tips helping to apply the rules. Figure 2.3 shows the simple LeSS process including some additional practices and concepts that are helpful for the application of LeSS.

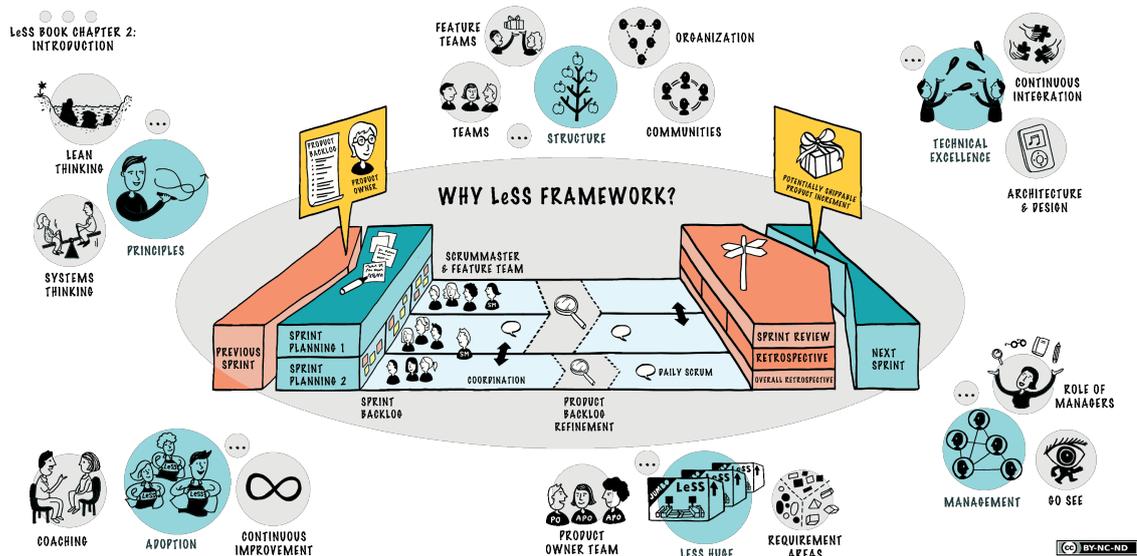


Figure 2.3: Overview of Large-Scale Scrum including roles, processes and artifacts [21]

The LeSS principles support organizations with their decisions on how to apply LeSS [55]. The first principle says that LeSS is Scrum meaning that the framework’s objective is to apply Scrum in large-scale contexts in a simple way (1). The second principle is transparency arising from tangible items, such as implemented functions (2). Thirdly, no further roles are added to keep responsibility in the teams, no additional artifacts are defined because the teams should focus on what they provide for the customer and no new process steps are defined to provide more space for situational learning and keep the process ownership to the team (3, “More with LeSS” [55, 21]). The focus is supposed to lie on the whole product which is reached by a single product backlog and one product owner responsible for it, one releasable product and all teams in the same sprint (4). Another principle is customer centricity. The goal is to provide a valuable product to the customer which can be only reached if the customer’s problem is understood and solved (5). Furthermore, the product has to be improved continuously (6). Lean-thinking should be applied by all managers. These pose frequent challenges to the teams, have respect for others, foster improvement and are teachers to the team (7). System-thinking supports the understanding and optimization of the entire product (8). Empirical process control means not to follow some predefined best practices, but to inspect and adapt not only the product but

also processes, practices and design continuously while focusing on the current situation and its challenges (9). Finally, it has to be understood how systems with queues function and apply those learnings to manage work-in-progress, variability and work packages (10) [55, 21].

In the next sections, the roles, processes, artifacts as well as tools and practices in LeSS are described followed by additions which need to be made if more than eight teams are involved and therefore LeSS Huge is needed.

2.4.2 Roles

The roles in LeSS are the same as in Scrum. The roles include one product owner for the entire product, Scrum masters, that are responsible for one to three teams each, and two to eight development teams. Conceptually, the role of the product owner has the same responsibilities as in one-team Scrum. However, next to managing the single product backlog, the product owner has to keep an overview of all teams and the entire product. The product owner has to ensure that the Return on Investment is maximized. This supports the required whole product focus and avoids the optimization of sub-products developed by sub-teams. While in Scrum the product owner focuses equally on prioritizing and clarifying backlog items, in LeSS his main responsibility is the prioritization. Clarification is mostly done by the teams in cooperation with the customer. Here the product owner is a connector between the teams and the customer. When the teams support the clarifications in that way, the product owner is able to focus more on the big picture and to explore further strategic opportunities. Further, the product owner communicates with higher level management to visualize the development progress, e.g. using burndown charts, and to adapt organizational design. The managers help to improve organizational capabilities and make decisions about organizational structures and policies [24]. The product owner cooperates with the Scrum masters who educate the product owner and help them to improve methodically. In a LeSS structure with many teams, the single product owner can be supported by other product managers, composing a product owner Team [23]. "Product owner is not a new name for a traditional project manager who delivers a scope and date contract of work. Rather, a product owner must have the independent authority to choose and change content, release dates, priorities, vision, etc. Of course she collaborates with stakeholders, but a real product owner has final decision-making authority" [23].

Scrum masters have a deep understanding of LeSS and coach the teams, the product owner as well as the organization about development practices and create a working environment in which the teams succeed [25]. One Scrum master can be responsible for up to three teams [21]. However, coordination between the teams is not the Scrum masters' responsibility.

The teams themselves are responsible for inter-team coordination [25]. The teams are self-managing, cross-functional and work across different components. The teams cooperate in a shared code environment. They are dedicated to take shared ownership and responsibility for their actions and team members use all their time to create a successful product.

Each team is co-located meaning that they work together in the same room to allow frequent face-to-face communication, team learning and to create trust and a sense of shared responsibility. Further, the teams need to be long-lived. Therefore, they should be stable over time to be able to cooperate closely and improve together over time [26]. If the teams are not able to create a shippable increment at the end of each sprint and have some open tasks left, an “undone” department exists which finishes those tasks before the release. Especially in smaller adoptions of LeSS, such a department should not exist, because the teams should be able to complete all tasks for the release. In LeSS Huge adoptions, sometimes an “undone” department can be useful for testing, Quality Assurance or architecture [22].

In LeSS Huge, teams are structured in requirement areas with four to eight teams each. The requirement areas are grouped around important areas of customer matters to ensure customer-centricity. Requirement areas are dynamic constructs whose size and priority varies over time. Each requirement area has an own product owner, the so-called area product owner who is responsible for the area product backlog. Sometimes the overall product owner can additionally be an area product owner [21].

2.4.3 Process

Most events in the LeSS development process are similar to Scrum events with some adaptations. All teams in LeSS are in the same sprint, that means in a sprint with the same start and end date as well as the same sprint goal. This allows for easy coordination of events which include representative of all teams. Coordination is also simplified by the co-location of the teams and can be supported by open spaces and communities though most communication takes place face-to-face or via code [21].

The sprint planning is divided into two parts. The sprint planning one is executed by the product owner and representatives of all teams. They discuss shared work and related backlog items. Further, it is decided which product backlog items will be implemented by which team. In the sprint planning two which is held in parallel by each team with all team members taking part the teams discuss design issues and make a plan for the upcoming sprint which is visualized by the sprint backlog [55]. If there is a lot of coordination required between some of the teams for the items in the upcoming sprint, they might hold their sprint planning two in the same room in different areas but with the possibility to communicate easily and frequently during the meeting.

There is no overall daily Scrum. Instead each team holds its own daily Scrum with the possibility of team members from other teams taking part to share information.

An overall backlog refinement with the product owner and representatives from all teams is a chance to align the teams and to discuss which team could implement which backlog item and to select those for discussion in the product backlog refinement in the affected team. The overall backlog refinement is only held if necessary [21]. The product backlog refinement held by each team separately has the same purposes as the corresponding Scrum event. It is used to refine the items the team is likely to implement in later sprints

[55]. This event can take place for several teams in the same room to simplify coordination and information sharing.

In the sprint review the product owner, representatives from all teams and customers as well as other stakeholders take part. The setting of the sprint review differs from the event in Scrum. It is meant to be more like a "science fair" with different areas where team members present those new functions their team has developed. The customers and stakeholders are shown the new items and are able to give feedback [21].

First, each individual team holds its own retrospective followed by the overall retrospective. In the overall retrospective, the product owner, the Scrum masters and representatives from all teams explore possibilities to improve the overall work [21]. Design workshops which incorporate agile modeling techniques and entire feature teams or members from different teams can be useful to determine design and architecture of items which need to be implemented. The design workshops are described more detailed in the Section 2.4.5.

In LeSS Huge, each requirement area conducts its own sprint planning. The overall backlog refinement is held for each requirement area instead of for the whole product. In order not to lose the whole product focus which is required there can be an additional product owner meeting with all requirement area product owners and the overall product owner to align and discuss overall issues. There are no rules for sprint reviews and retrospectives in LeSS Huge. Certainly, there are reviews and retrospectives for each requirement area. Additionally, there can be a review and retrospective for the whole product if considered necessary [55]. The further events remain the same as for LeSS [21].

2.4.4 Artifacts

The artifacts in LeSS are an overall product backlog, separate sprint backlogs for each team and one integrated product increment. The product backlog does not differ from the corresponding artifact in the one-team Scrum. It therefore includes prioritized and estimated items ordered by their granularity with more detailed items at the top. The sprint backlogs consists of the product backlog items and all tasks to implement those a team needs to complete in the current sprint. It is the same as in one-team Scrum. At the end of each sprint an integrated and potentially shippable product increment is finished. The integration itself is done during the sprint. This supports transparency and the whole product focus, reduces risk and work in progress and keeps feedback cycles short to allow for continuous improvement. Further, a common definition of "done" which is initially defined and agreed on but evolves over time [21].

The application of LeSS Huge requires a requirement area attribute in the overall product backlog to visualize which requirement area works on the item. As a result, those backlog items with the same area attribute can be grouped in the so-called requirement area backlog [21].

2.4.5 LeSS and Architecture

"We believe that agile architecture comes from the behavior of agile architecting. It is primarily about mindset and actions, not the use of a particular design pattern. And part of that mindset is thinking 'growing' or 'gardening' over 'architecting'." [52]

Craig Larman and Bas Vodde, the creators of LeSS, consider the term "architecture" which was borrowed from actual building architects a bad analogy as it implies that architecture has to be done before starting to build. Once the construction of a building started it is nearly impossible to change its architecture. On the other hand, software and accordingly its architecture are not supposed to be fixed permanently but still changeable over time, especially when agile methods are used. Architecture changes or "grows" over time. Larman and Vodde are critical about defining an intended architecture at the beginning of the development process as this is often only done in documents. Later, this intended architecture might not correspond to the actual architecture in the system and even might not be the best suitable one as well. Further, they criticize that often architecture and the source code are considered to be completely separate from each other [54]. Larman and Vodde present some observations they made about agile architecture and design [54, 20]. Their first observation is that the source code is the true architecture or design of the software. The architecture defined in the source code often does not correspond to the architecture which is intended by the team: "The architecture is what is, not what one wishes it to be" [20]. Based on the first observation, the second one is that architecture evolves over time with every line of code which is written. Architecture is not a static construct but changes continuously as long as the development team is programming. Additionally, Larman and Vodde observed that "the real living architecture needs to be grown every day through acts of programming by master programmers" [54]. In general, the LeSS framework assumes that architecture is rather grown than being built. Further, the programmers themselves are the ones to determine the true architecture. This can be connected to the fourth assumption which says that an architect that does not know or work on "the evolving source code of the product is out of touch with reality" [20].

In LeSS, the role of the architect is not defined as one person in the team, but every member of the development team is an architect, even if he/she does not know or want it. As soon as a developer changes the code or adds something to it, it is an architectural act [54, 20]. In large organizations often those observations are not acknowledged and architecting is exclusively done by architects outside of the development teams most likely before the development process starts. Agile architecture arises from the team's behavior concerning architecture including agile mindsets or culture as well as agile practices and actions [52]. Larman and Vodde suggest "hands-on master-programmer architects, a culture of excellence in code, an emphasis on pair-programming coaching for high-quality code/design, agile modeling design workshops, test-driven development and refactoring, and other hands-on-the-code behaviors" [54, 20]. Therefore, they mainly give behavior oriented tips and only few technical oriented tips when it comes to architecture and de-

sign in LeSS [54, 20].

As an important step for improving architecture or designing new items to be implemented Larman and Vodde suggest to hold design workshops for agile modeling. These workshops involve the entire feature team which is responsible for building the new item to be modeled and designed. A team might do a design workshop right before starting to build the item or whenever the team considers modeling useful. Most requirements should be clarified before the workshop though during the modeling new questions may arise. The feature team decides what to model and depending on the team's objective they can model only the new item or the overall system architecture [54, 20]. The team decides which kind of model is useful for reaching their objectives: "low-fidelity UI modeling with sticky notes or in prototyping tools, algorithm modeling with UML activity diagrams, object-oriented software design modeling usually sketched in UML-ish notation, and database modeling likewise" [20]. The authors suggest modeling on large whiteboards and even covering the entire wall of the team room with posters or whiteboard-like material which sticks to the walls to be able to make sketches that the whole group can see and can get involved in. It helps the team to model in a simple, graphic way. They see those models not as specifications but as a basis for having constructive discussions about the new item ("Model to have a conversation" [20]). LeSS acknowledges that all models are "wrong" and that they evolve. They serve as an inspiration and help the team to learn and reach a shared understanding.

It is useful to hold modeling workshops in their team room every iteration. Such workshops should be time-boxed as every Scrum event, but can last from two hours to two days. If design or architecture issues affect more than one team, those teams should take part in the workshop as well. Larman and Vodde also encourage LeSS teams to involve modeling in their daily work. If developers are stuck during programming or have any questions, they could sketch a simple model and invite their team members for a short discussion to get their opinions or a new perspective on their issue. Therefore, team members are able to translate code into simple or very detailed models and the other way around - whatever abstraction level is required. Further, architects can develop spikes solution to refine requirement. Spikes are very small solutions going vertically through all components to explore suitable architecture for a new feature.

In the beginning of new product development or if major changes to the architecture are necessary it makes sense to establish a small co-located tiger team of architects, that are programmers at the same time. The tiger team explores architecturally significant features and implements them. As soon as the tiger team gets too small to handle all challenges, it is divided. Then the tiger team members become part of different new teams which they can support with their gathered knowledge and skills. A system architecture documentation workshop helps to spread the knowledge gained by the tiger team and create a shared understanding for all teams. The former tiger team members can coach the teams on architecture and programming practices and can take on the role of technical leaders in the teams. Additionally, Communities of Practice (CoPs) can bring together technical leaders and programmer-architects from different teams to work on common (architectural) issues

and share knowledge. CoPs can cooperate and document their findings in a shared Wiki Space for themselves and their teams. If the code of some components is shared, the role of a component mentor who knows a certain component in detail, teaches and informs others about the component and ensures that the changes made by the feature teams on the component make sense can be helpful [54, 20].

As Scrum teams and LeSS teams are cross-functional they should be able to provide their customers with entire solutions. Hence, there is no separate architecture team, but architects should be included in the teams as regular members. Architectural tasks are part of the product backlog and work is done in the sprints. The requirements which have a major impact on the architecture should be implemented early to minimize the risks of major architectural changes later in the development process. The architecture evolves over time and in the process all earlier architecture decisions should be questioned and - if necessary - changes should be made. However, some early analysis and decisions are required, such as choosing a programming language. Especially if some architectural decisions seem to be outdated, teams should make the effort to find a better technical solution and to convince others that a change might bring benefits later on. If earlier decisions are questioned and continuous actions for improvement are taken, technical debts can be reduced. This is especially important if teams develop a solution which now may be little, but is supposed to become really large. Re-factoring and continuous integration are essential. Additionally, one should avoid "architecture astronauts", "Powerpoint architects" or ivory-tower architects that are out of touch with the code and therefore the actual architecture [20]. In large organizations in which a separate architecture team exists, the role of the architect needs to change. These architects can take on the role of teachers or mentors to the team and provide the teams with methodical competences or new practices as well as tools and can be a connector to the overall business vision. However, Larman and Vodde prefer architects who are developers and technical leaders at the same time, know the code, provide guidance during pair programming or design workshops, push the usage of new technologies, make architectural decision and communicate them [54, 20].

Finally, next to other methods Larman and Vodde recommend the use of DDD as it "encourages thoughtful iterative design, shared understanding, and a domain model that must be well-expressed in the code" [54]. DDD can give the teams helpful tools for domain modeling. DDD can support the teams to reach a shared understanding of their domains by providing the advice that all code reflects the domain model and that the model is the basis for a shared language for all team members. Though this is mostly related to tactical DDD, Strategic DDD can support the definition of the team organization according to business domains. It is suggested that the teams are structured around customer requirements and not around architecture. Therefore, the feature teams are the preferable team composition to be able to develop "vertical" features fulfilling the customer's requirements. Feature teams develop and maintain features end-to-end [4]. This means that the teams program to fulfill a customer requirement by implementing the feature across all layers and components [54, 20]. Verticalization can be supported by Strategic DDD 2.7.4.

2.5 Scaled Agile Framework (SAFe)

The following section gives an overview of the Scaled Agile Framework (SAFe 4.0) with its most essential objectives, roles, the process, artifacts and the role of architecture for agile scaling.

2.5.1 Framework and Objectives

The Scaled Agile Framework was first published under this name by Dean Leffingwell in 2011. In the beginning of 2017 (until June), the most current version was SAFe 4.0. Based on agile and lean principles, the Scaled Agile Framework (SAFe) can be applied in large organizations with many agile, sometimes geographically distributed teams to develop complex enterprise systems [38]. SAFe helps to align numerous teams and can be adapted to the needs of different organizations, as it is modular and can be applied in different ways to provide better business outcomes [40]. SAFe is based on the three main bodies of knowledge agile software or systems development (see Section 2.1.1), lean product development and flow or systems thinking [58]. SAFe is not a predefined process which can be applied step by step by each organization which wants to become more agile. SAFe provides a starting point, patterns and a knowledge base. Therefore, nearly all of the described elements are optional. The actual implementation might differ for each organization [63].

Two general bigger pictures of SAFe exist. The three-level view is suitable for a modest number of teams with up to 100 practitioners and includes the team, program and portfolio level as well as some foundations. The four-level view comprises additionally a value stream level and is suited for projects which require more than hundred team members to develop large and complex solutions [58]. While SAFe is the most complex of the scaling methods presented here, it can be used to make an entire enterprise or large parts of an organization more agile. Figure 2.4 presents four-level SAFe with all its roles, processes and artifacts.

2.5.2 Roles, Process and Artifacts

SAFe consists depending on the size of the enterprise or project of three or four levels: the team, the program, the optional value stream and the portfolio level. The different levels are described in the following paragraphs.

Team Level On the team level SAFe includes agile teams of which each is responsible for specifying, implementing and testing user stories from their own team backlog in a series of iterations with predefined duration (like sprints) [40, 57]. Each team has between five and nine members and applies either Scrum or Kanban methodology with lean product development practices and eXtreme Programming elements. In this work, the focus lies on Scrum extended by some lean and eXtreme programming practices. The roles within

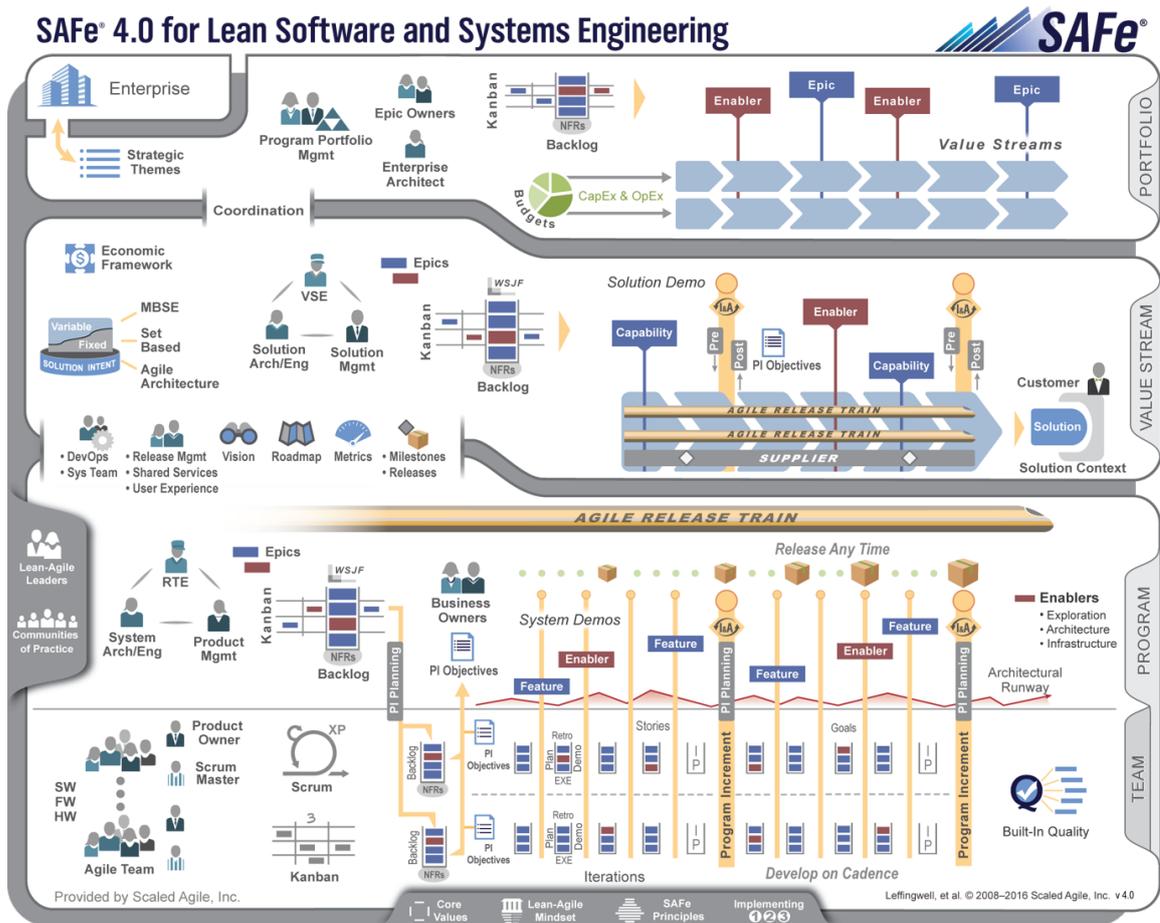


Figure 2.4: Overview of four-level SAFe including roles, process and artifacts [43]

each of the teams are Scrum master, product owner as well as developers with responsibilities as defined by Scrum. The product owners maintain and refine the team backlogs with their team’s user stories and are further responsible to plan the product increments and value to be provided in cooperation with product management [40].

While in small organizations only a few teams exist, in larger enterprises groups of teams cooperate to build up extensive functionality into final products, features, subsystems or architectural components [57]. In larger organizations, the teams should follow a standard iteration length, also known as cadence, with the same start and end time so that the new functionality can be integrated at this point. For each iteration the same process is followed by each of the teams. First, they plan the iteration and commit to functionality to be implemented. After that the development itself and testing takes place. Finally, similar to a sprint review, new functionality is demonstrated in the team demo and a retrospec-

tive is held (iteration retro) [58]. Daily stand-ups, similar to daily Scrums, are conducted by the teams internally. The development timeline is not only divided in iterations, but also includes the program increment (PI) planning which is a meeting lasting two days and taking place every eight to twelve weeks which is used to commit to a set of common goals until the next PI. The PI requires a lot of preparation including the definition of a vision, milestones and a roadmap in cooperation of the product management, agile teams and the customer. The PI is the most essential event to communicate with the program level [40].

Further artifacts correspond to Scrum methodology and are sprint backlogs for each team, user stories, a definition of "done" and the integrated increment [57]. Quality of the software developed by a team is assured by the team members themselves. However, additional quality assurance personnel is responsible for system-level testing to ensure system-level quality. Further, the teams can be supported by user-experience and data-base designers, documentation as well as build and deployment specialists and whomever else is suitable to develop a successful integrated product or shared services. Even if agile teams do not include the role of an architect, architecture matters. The architecture is determined by the local teams and emerges from the teams' activities [57].

Program Level At the program level, five to twelve agile teams are organized into a so-called "agile release train" (ART). An ART is a long-lived and self-organizing virtual program structure which is to deliver value in the form of PIs to the enterprise's most essential value streams [40]. Business owners, a critical group of three to five individuals, share "fiduciary, governance, efficacy, and ROI responsibility for the value delivered by a specific Agile Release Train. For each train, the role is fulfilled by the people best suited to these responsibilities" [58]. Next to the teams organized in the ARTs, on the program level additional roles exist as some further higher level objectives need to be reached [40]. They are also responsible for the program backlog.

The objectives at the program level are maintaining the vision and the roadmap, coordinating releases, assuring quality of the aggregated results of each ART, managing deployment for the end users, managing resources and eliminate those impediments that hinder the teams to reach their goals which are outside of the teams' control [57]. The three additional roles on the program level are release train engineer, product management and system architect/engineer. The release train engineer as the chief Scrum master of the ART optimizes continuous value delivery by using PI Planning. He organizes a Scrum of Scrums meeting taking place weekly or even more often with the Scrum masters to make progress and any impediments transparent [40]. Product management cooperates with the customer and product owners to understand the customer's requirements and define them in the program backlog for the ART for which the Product Management is responsible [58]. Product management meets up with product owners in the "product owner sync" to get an insight into the progress of the ART concerning the PI objectives on the program level [40]. The system architect/engineer who is either one person or a small

cross-discipline team define the overall architecture of the large-scale system, define non-functional requirements and interfaces [58].

Another concept included on the program level is the architectural runway symbolizing the emergence of architecture over time including not only emergent design, but also intentional architecture which is required to guide the teams as soon as the number of agile teams and the size of the system grow larger. Further information on the architectural runway is given in Section 2.5.4 along with information about agile architecture in SAFe.

At the program level, one has to decide if the teams in the ART should be organized as component or feature teams to optimize value delivery and simplify integration. Component teams are organized around specific components of the architecture of a large-scale system. For example, there might be a team for each layer of the architectural stack, such as one team for the presentation layer, one for the business logic and one for the database. In contrast to component teams, feature teams are organized around certain features, meaning that a team develops a feature including not only the business logic, but also the presentation layer and the database. The core competence of each team is one feature or a set of similar features and not only one part of the technology stack. Many sources find feature teams preferable even if it means a higher effort for reorganizing the enterprise. Advantages of feature teams include that the team builds up expertise in its domain, ability to develop end-to-end features, fewer interdependency between the teams, less overhead in the product backlog and increased value throughput [57].

Between the program and the value stream Level the so-called "Spanning Palette" is presented which includes roles and artifacts which can apply on different levels of SAFe, though they are most common in the program and value stream level. The "Spanning Palette" includes amongst others vision, roadmap, milestones and DevOps [40]. Either on the program or value stream level inspect and adapt Workshops take place at the end of each PI to identify possibilities to improve the next PI concerning reliability, quality and velocity [58].

Value Stream Level The value stream level is optional and only necessary if extremely large and complex systems are developed. While for simpler solutions only one ART exists on the Program level, for more complex solutions many synchronized ARTs are needed. Each value stream serves to synchronize multiple ARTs. The cadence on the value stream level is based on ART PIs to be able to synchronize the ARTs. Some further roles, such as value stream engineer, solution management and solution architect/engineer are necessary [40]. As the servant leader of the value stream, the value stream engineer is responsible for identifying and resolving bottlenecks across the value stream. The solution management is responsible for the value stream backlog representing the customer's requirements and the capabilities to be implemented as well as the overall strategy defined by the portfolio level and for governing the economic aspects of the decision-making for the ARTs. The solution architect/engineer specifies the overarching architecture to connect the solution across the various ARTs. Additionally, they guide the system architects/engineers

in the ARTs [58]. As in the case study in this thesis the systems might not be as large and complex, the value stream level will not be needed if the enterprise decides to apply a variation of SAFe. Therefore, the process, roles and artifacts of this level are not explained in detail. However, the value stream level includes some interesting aspects when it comes to architecture in SAFe. These aspects are explained in Section 2.5.4 .

Portfolio Level The top level in SAFe is the portfolio level which either communicates with the value stream level, if it exists, or directly with the program level. The portfolio level serves to organize and fund one or more value streams of which each develops a solution to reach the enterprise's strategic goals. The portfolio level provides strategic themes, coordination of the value streams, lean-agile budgeting and governance [40]. The portfolio level connects the whole framework to the enterprise by defining strategic themes in cooperation with other key stakeholders in the enterprise. Strategic themes are the business objectives derived from the overall enterprise strategy. Further, the portfolio level provides feedback to the enterprise, for example key performance indicators (KPIs), information about the current solution's fitness and current strengths and weaknesses as well as possible opportunities and threats [58]. The program portfolio management (PPM) is one of the key roles on this level considering all knowledge about the enterprise strategy, financial and technological constraints. PPM is responsible for any investments and returns as well as for any content to be developed.

Major artifacts are portfolio epics which are large initiatives required by the enterprise and including all business capabilities that need to be developed or improved [40]. Portfolio epics first go through the portfolio Kanban to be analyzed and approved for implementation considering strategic initiatives and economic aspects. The portfolio epics which are approved to be implemented are organized in the portfolio backlog and are managed by the epic owners [57]. One can differentiate between business epics and epic enablers on the portfolio level. Business epics comprise impact or cost analyses or business cases connected to potentially evolving business capabilities to be approved before deciding if it should be implemented. Enabler epics include architectural or technical topics which serve to enable the new business capabilities. The flow of work on the portfolio level is made transparent by a portfolio Kanban system. The enterprise architect role requires knowledge about all value streams and programs to be able to provide strategic technical guidance. The enterprise architect often is the owner of enabler epics with the objectives to harmonize delivery and development [58]. In Section 2.5.4, the architectural runway and the enterprise architect's role are described in detail.

Foundation Layer Additionally, SAFe includes a Foundation Layer which holds practices, values and principles to support large scale development, such as core values, Lean-Agile mindset, lean-agile principles, lean-agile leaders and CoPs). The core values are alignment, transparency, built-in-quality and program execution. The Lean-Agile mindset includes the values as defined in the Agile Manifesto plus lean principles, such as respect

for people and culture, flow innovation and relentless improvement. The lean-agile principles in SAFe are to take an economic view, apply systems thinking, assume variability and preserve options, build incrementally with fast, integrated learning cycles, base milestones on objective evaluation of working systems, visualize and limit work-in-process, reduce batch sizes, and manage queue lengths, apply cadence, synchronize with cross-domain planning, unlock the intrinsic motivation of knowledge workers and decentralize decision-making [40]. Lean-agile leaders follow all these principles while being life-long learners and teachers to their teams. CoPs are established to allow a group of experts and members from different teams to discover, share and discuss practical knowledge [58].

2.5.3 SAFe Update

During writing this thesis in June 2017, an update of SAFe was published (SAFe 4.5). In this section, the major differences are listed and discussed shortly. Though most roles stayed the same, there are now four different versions of SAFe for different organizational sizes and complexities. However, SAFe still has to be configured differently to fit different organizations. The four drafts now may come closer to the solution the organizations really implement. The four configurations are Full SAFe, Portfolio SAFe, Large Solution SAFe and Essential SAFe. Essential SAFe describes the basic elements of SAFe on the team and program level. Portfolio SAFe corresponds to the former three-level SAFe. Large Solution SAFe is for more complex organizations which require multiple ARTs. This configuration of SAFe consists of Essential SAFe plus a large solution level (former value stream level), while it does not include the portfolio level. Full SAFe then also includes the portfolio level. The poster for the four different configurations can be found on the website of Scaled Agile Inc [44].

What previously was a value stream is now the solution train which serves to manage several ARTs. Additionally, compliance is now part of SAFe and is considered as a part of the solution intent. Agile architecture was removed from the big picture, however it is still a very important component in the guidance articles and a part of the solution intent. Based on the lean start-up cycle, now also minimum viable products and lean business cases are part of SAFe. Additionally, DevOps and continuous delivery play a role in the ARTs [44].

2.5.4 SAFe and Architecture

SAFe underlies the assumption that architecture is agile. However, opposing to Nexus and LeSS, in SAFe not all architecture is said to emerge purely through the act of programming. Leffingwell calls this component emergent design, while additionally he says a part of the architecture is intentional and is created in collaboration of architects, technical leaders and the teams [40, 56]. However, in SAFe one should avoid big up-front design and processes with plenty of interruptions. Additionally, the system is supposed to run always and value has to be delivered continuously. Emergent design ensures a suitable technical

basis for iterative development and incremental delivery and the possibility to adapt the design according to customer needs quickly. Corresponding to the approach to architecture in Nexus and LeSS a part of the architecture emerges along with programming and deploying new features. However, according to Leffingwell large enterprises face further challenges which can only be coped with successfully with large-scale architectural initiatives. These initiatives require certain objectives (or "intentions") and more planning. Large-scale architectural initiatives address overarching technical issues and provide guidance to agile teams on the team and program level. They help to reduce redundancy, guide the teams concerning approaches and want to reach a higher reliability and robustness of the overall system [41, 58]. Both, emergent design and intentional architecture contribute to the architectural runway which is shown in Figure 2.5.

The architectural runway was defined by Leffingwell in 2007 as follows: "A system that has architectural runway contains existing or planned infrastructure sufficient to allow incorporation of current and anticipated requirements without excessive refactoring" [56]. Therefore, in the agile enterprise context with many incremental releases the architectural runway means to answer the question: "What technology initiatives need to be underway now so that we can reliably deliver a new class of features in the next year or so?" [57]. These initiatives contain significant architectural changes, possibly affecting all teams. The implementation of those large-scale structural changes are planned, estimated and organized as architectural epics on the portfolio level. Architectural epics help to reach the enterprise's technology vision and often require high investments and a lot of resources. They bear the risk to reduce the current velocity, but may be necessary to increase the velocity and quality of the system in the future. On the program level architectural epics are divided into architectural features for the upcoming release meaning that they have to be done on time for the release. They are handled like all other features on the program level. The development teams contribute to the architectural runway through re-factoring and by conducting spikes. These actions are handled like any other user story in the team backlogs and must be demonstrated at the end of each iteration [57].

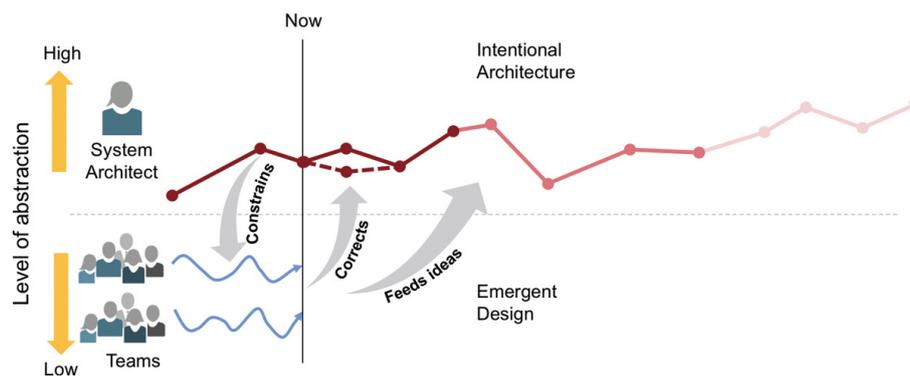


Figure 2.5: Architectural runway in SAFe [41]

Agile Architecture in SAFe is defined by a set of principles. The principles by Leffingwell have evolved over time and some differ in the various sources [59, 57, 41, 58]. The first principle is that design emerges and that architecture evolves by collaboration of persons on all levels (1). This principle was adopted from the 11th principle of the Agile Manifesto: "the best architectures, requirements, and design emerge from self-organizing teams" which is the central assumption for architecture in Scrum [3, 58]. SAFe acknowledges that emergent design is very effective at the team level, but that on the other levels some additional guidance is necessary. Intentional architecture provides guidance for issues concerning more than one team and the teams' interdependencies and helps to improve usability and performance of the overall system. It is done in a more abstract and less detailed way to leave space for the teams to adopt the constraints and ideas in their context. Though intentional architecture sets constraints for the emergent design, the team level design can contribute to intentional architecture with new ideas which might deliver value to other teams in the future as well. So if one team found a good solution to an issue other teams may also face in the future, this solution can contribute to intentional architecture and other teams can profit from it as well. The collaboration for evolving a suitable architecture involves not only the development teams, but also system, solution and enterprise architects as well as product and solution management. Summing up, in a very complex environment a global view on architecture is required to anticipate changes to the overall system and to ensure integrity, synchronization, performance, scalability and maintainability [41, 58].

The second principle is "The bigger the system, the longer the runway" (2) [41]. Large teams working on large systems need to improve the architecture continuously to also be able to deliver features which are required later in the project on time. Therefore, a suitable technological infrastructure is necessary to implement new features to avoid excessive re-design which causes - if realized too late - high delays. While intentional architecture contributes to the architectural runway by laying the foundations for the implementation of high priority features in the future, emergent design is done by the agile teams developing features in the current iteration. Building the runway for large systems takes a lot of time and is a continuous process which is an essential part of the project [57, 41, 58].

Thirdly, the teams are given advice to "build the simplest architecture that can possibly work" (3) [41]. In agile development projects requirements changes over time. To enable those changes in the developed product, the architecture has to be kept as simple as possible while still ensuring the current functionality [57]. Simplicity is supported by establishing a common language, refactoring, keeping the solution close to the problem domain, ensuring that the intent of interfaces is expressed transparently and by following design principles. Relating to this principle, Leffingwell suggests DDD to keep design and communication simple by fostering feature-orientation [41, 58].

"When in doubt, code or model it out" is another principle of agile architecture in SAFe (4). To explore how a new item should be implemented, the teams can just code it out, use spikes or even rapid prototyping to find a suitable solution. If this is not enough, modeling can help to understand the problem to be solved and understand its impact. Especially,

domain modeling can be useful to gain a better understanding of the entities in the problem domain and their relationships. Therefore, Leffingwell suggests: "If you model one thing in agile, model the domain" [42]. Domain modeling not only supports the analysis of requirements, but can also help to determine the conceptual design. Here, the DDD approach can help to model, design and implement new features and to manage complexity [42, 41].

The next principle says that the ones who design and build a system and its architecture are also responsible for testing it (5). The development teams have to be able to test all functional and non-functional requirements they have implemented continuously [57, 41, 58]. Another principle is that "There is no monopoly on innovation" [41]. There is not a single person, role or team who is responsible for innovation, but all teams, architects as well as engineers and all stakeholders can contribute to the system with their innovative ideas. Agility in SAFe also means to foster a culture of innovation [57, 41, 58].

The last principle is to "implement architectural flow" by coordinating and implementing large-scale architecture initiatives on all levels [41]. This can only be reached by improving continuously and making necessary changes visible and transparent [57, 41, 58].

Summing up, those principles are central to architecture in SAFe and DDD can support the teams by making the underlying logic explicit by using models, by providing a basis for a shared language, refactoring and linking the domain and solutions to issues in the domain [41, 58].

2.6 Comparison of Nexus, LeSS and SAFe

To summarize the main ideas, processes, roles, artifacts and architecture in Nexus, LeSS and SAFe the next sections include a comparison of the frameworks. For the comparison some criteria from the Agile Scaling Knowledge Matrix that compares many approaches were used in an adapted ways [71]. However, many new and more detailed criteria were added. The results then are reused to find a suitable approach for the large insurance enterprise in the case study which is the goal of the third research question. Differences in architecture in the frameworks and how DDD contributes to architecture, are explored in sections 2.6.5 and 3.2.1.

2.6.1 Frameworks and Objectives

The results of the comparison of the objectives of the three frameworks are presented in Table 2.4. All of the three frameworks are available in two versions. While Nexus supports agile scaling for up to nine Scrum teams, Nexus+ suggests that several Nexus units which should be as independent as possible from each other can be coordinated using Nexus+. However, there is no detailed information about how coordination happens in the framework [78]. LeSS scales up to eight Scrum teams, while LeSS Huge scales up to a few thousand individuals, if all teams are organized in requirement areas. In SAFe, the

number of teams that can be scaled depends on if the three or four level view is used. By adding the optional value stream level hundreds or even more people can be scaled [58]. Whereas SAFe, tries to make very large parts of enterprises more agile and to cover all levels of the organization from team level to portfolio level, in Nexus and LeSS the focus is more on the integration between the teams [17]. In Nexus, the higher levels are not really specified. However, in Nexus the integration team might function as a connector to higher level management. Similarly, in LeSS the product owner (referred to as PO in the tables) connects the teams to management. Differences also occur in inter-team coordination. While in Nexus the coordination is done by the integration team, in SAFe the product owner and Scrum master are responsible for coordination in exchange with the program level and in LeSS the teams themselves and the product owner are responsible for inter-team coordination.

In all frameworks there are joint events with people from different teams and with overarching roles that support inter-team coordination [14]. The objectives of the frameworks are quite different. SAFe supports even very large organizations to become more agile by providing a set of practices that are all optional and adaptable to any organization [2]. However, it is often argued that SAFe is not really agile including many top-down decisions and a lot dependencies that potentially slows development down. Further, a misalignment with Scrum is stated especially because of the product owner roles which potentially lead to forgetting that whole product focus is essential [82]. Schwaber criticizes SAFe and other large scaling frameworks by saying that they "started to reintroduce methodological, prescriptive, one-size-fits-all thinking back into software development, at the same time claiming that it was agile. However, all of these approaches fall apart at the software development level. They typically just say "'use Scrum', but Scrum is a one-team, one-sprint thing. Also, these methodologies do not tell you how to run large, multi-team Scrum projects or programs" [78]. Schwaber who is one of the founders of Nexus therefore wanted to define a framework which is simple and respects the central values of Scrum. This is done by keeping and slightly adapting the Scrum roles and adding the Nexus integration team on top to reach what one-team Scrum is missing: a way for coordinating many teams and integrating their work [78].

However, as Nexus is the newest of the discussed frameworks, there are not many case studies or success stories of companies that used Nexus and there is not much information that is available to the public. Therefore, the implementation of Nexus can be very difficult without sending many people to Nexus/ Scaled Professional Scrum workshops [76, 60]. Similar to Nexus, LeSS is considered to be really agile [16]. Especially for small organizations, that grow larger the approaches in LeSS seem to feel quite natural. However, for large organizations large organizational and architectural redesign might be required [16, 2]. Additionally, Scrum teams in LeSS should be experienced in using Scrum methodology to be able to deal with not only the development, but also inter-team coordination and emergent architecture [82].

	Nexus / Scaled Professional Scrum Schwaber	Large Scale Scrum (LeSS) Larman/Vodde	Scaled Agile Framework (SAFe) Leffingwell
Number of Scrum Teams	Nexus: three to nine Scrum teams Nexus+: several Nexus units exist, which are as independent from each other as possible, no exact number of teams defined	LeSS: Up to eight Scrum teams LeSS Huge: more than eight teams up to a few thousand people	3-level view: for smaller projects of around 100 people or less 4-level view: for large solutions of hundreds or even more people
Coverage of all "levels"	- Focuses on the integration role between the teams - does not cover the higher levels in organizations	- Focus on the integration of several teams - no levels above, PO is connector to higher levels	- covers all levels - aim to even make a whole enterprises more agile
Inter-Team Coordination	- done by Nexus Integration Team - joint events, such as Nexus Daily Scrum, Sprint Planning, Retro	- done by the teams - joint events, such as overall Retrospective and Refinement and Sprint Planning One	- few details about coordination between the levels - joint events, such as PI Planning, Scrum-of-Scrums
Main Objective	- simple approach to Scrum for larger projects	- "More with LeSS" - less processes, less artifacts, less roles... - Keeping Scrum and its objectives as it is and use it in large-scale development - changing the organizational structure will change culture	- allowing large, traditional organization more agility - making the IT organization more agile on all levels
Criticisms	- quite new, so there are not many success stories available - not many resources and information available - no guidance on architecture, not even on team level	- too much agility for large, traditional organizations - teams need to be very mature concerning agility to be able to develop and to do inter-team coordination and integration	- not really agile, because it is too prescriptive - teams are told what to do top-down
Key strengths	- really agile - Focus on integration	- really agile - could feel very natural to small organizations as they grow larger	- very adaptable to the organizational needs

Table 2.4: Overview of the frameworks and their objectives (see Sections 2.3, 2.4, 2.5 for descriptions and further sources)

2.6.2 Roles

A comparison of the roles in the frameworks is shown in Table 3.2. The team structure and roles are largely the same as in Scrum. However, there are some additional roles and some changed responsibilities in the different frameworks. In Nexus and LeSS, a single product owner exists who is responsible for the overall product backlog and who has an overview of the whole product as well as the goals to achieve with the product. On the other hand, in SAFe every development team has an own product owner who manages the team backlog and focuses features implemented by the team. In all frameworks, a product owner prioritizes and clarifies backlog items. However, in LeSS, it is explicitly said that clarification is supported by the teams cooperating with the customer. Here, the product owner has the function of a connector between the teams and the customer. Additionally, he/she is the

connector to higher level management, e.g. concerning budgeting. To foster integration, the product owner is part of the Nexus integration team in Nexus and cooperates closely with product management in PI planning.

In all frameworks, there are Scrum masters who are responsible for one or more teams and mainly coaches the associated teams concerning the used methodologies and ensures their application. In Nexus, one of the Scrum masters is part of the Nexus integration team ensuring the overall application of Nexus and integration. In SAFe and Nexus, the Scrum masters play a bigger role in integration and cross-team coordination than in LeSS. In LeSS, the teams themselves are responsible for inter-team coordination. In SAFe, there is an additional release train engineer who takes on the responsibilities of a Scrum master for the ART.

In the three frameworks, all teams are feature teams. However, the Nexus guide does not say explicitly that teams are organized depending on features, but still says that teams are cross-functional, self-organizing and take responsibility for the features they deliver. The development teams support the clarification of user stories, as well as their estimation. They are responsible for implementing features and testing them. LeSS is the only framework in which no additional roles are added except if LeSS Huge and therefore Area product owners are required. In Nexus, the integration team is an additional role or construct for cross-team coordination and for ensuring overall integration of features developed by the teams into an integrated increment. As SAFe is used to make very large projects more agile and because of its level structure, many additional roles are required that are not comparable with any roles in the other frameworks [76, 21, 55, 40, 14, 58].

	Nexus / Scaled Professional Scrum Schwaber	Large Scale Scrum (LeSS) Larman/Vodde	Scaled Agile Framework (SAFe) Leffingwell
Role of the Product Owner	<ul style="list-style-type: none"> - single PO for all teams - manages the single Product Backlog - whole product focus - prioritization and clarification of Backlog Items - part of the Nexus Integration Team 	<ul style="list-style-type: none"> - single PO for all teams - manages the single Product Backlog - whole product focus - focuses on prioritization more than on clarification - connector between the teams and the customer - connector to higher level management 	<ul style="list-style-type: none"> - one PO per Team - manages the Team Backlog - focus on features built by the team - prioritization and clarification of Backlog Items - plans the PIs in cooperation with Product Management
Role of the Scrum Master	<ul style="list-style-type: none"> Scrum Masters (one per team) - responsible that Nexus is enacted in the teams - responsible for one/more of the team Integration Team Scrum Master (one per Nexus) - responsible that Nexus is enacted overall - can additionally be Scrum Master of one/more teams 	<ul style="list-style-type: none"> Scrum Masters (one per team) - deep understanding of LeSS and development practices - create a working environment in which the teams succeed. - coach the teams, the PO as well as the organization 	<ul style="list-style-type: none"> Scrum Masters (one per team) - coach the team concerning agile behavior and self-management - cross-team coordination and remove impediments - part of Scrum-of-Scrum meetings and reports to management Release Train Engineer (one per ART) - takes on the responsibilities of a Scrum Master for an ART
Teams	<ul style="list-style-type: none"> - know-how about different parts of the business/ computer systems - domain knowledge considered for team composition - cross-functional and self-managing - not necessarily feature teams, but assumptions imply it - improve user stories and acceptance criteria - implement and test user stories 	<ul style="list-style-type: none"> - self-managing, co-located and long-lived - cross-functional and cross-component feature teams - take shared ownership and responsibility - clarify, implement and test user stories 	<ul style="list-style-type: none"> - self-managing, accountable for the delivery of their features - cross-functional & cross-component feature teams - improve user stories and acceptance criteria - implement and test user stories
Additional Roles/ Constructs	<ul style="list-style-type: none"> Nexus Integration Team - the PO, a Scrum Master and further possibly changing members - responsible for the integration of all feature into a single PI - responsible for cross-team coordination - supervise the application of Nexus and the operation of Scrum - behaves like any other Scrum Team 	<ul style="list-style-type: none"> LeSS Huge: Area Product Owner - responsible for the Area Product Backlog 	<ul style="list-style-type: none"> - Program Level: Business Owner, Release Train Engineer, Product Management, System Engineer/Architect - Value Stream Level: Value Stream Engineer, Solution Management, Solution Architect/Engineer - Portfolio Level: Program Portfolio Management, Epic Owners, Enterprise Architect ...

Table 2.5: Roles in Nexus, LeSS and SAFe (see Sections 2.3, 2.4, 2.5 for descriptions and sources)

2.6.3 Process

A description of the central events can be found in Table 2.6. The events defined by Scrum are part of all frameworks with some adaptations. Especially in Nexus and LeSS, the names of the events correspond to Scrum events. In SAFe, which adds further events, some of the names of the events are adapted, such as iteration instead of sprint and team demo instead of sprint review [32].

For the inter-team coordination, the goals of some meetings are changed and some additional events are required. However, as in Scrum all work is done in sprints with fixed length. Sprints - or iteration in SAFe - last between one and four weeks. Thus, in the SAFe framework an iteration length of two weeks is suggested.

When it comes to refining the backlog, the frameworks differ. In Nexus, the backlog refinement happens not only continuously but also as a time-boxed event with representatives

from all teams who first decompose and detail items and second identify and visualize dependencies between the teams. Further, it should be identified as early as possible which team will implement an item so the team can get involved in the refinement. In LeSS, the overall backlog refinement takes place if necessary and also includes appropriate members of all teams whose main tasks are not only refining the items but also to discuss which team is suitable to implement a certain user story and to align the teams. Additionally, there is a backlog refinement for each team in which the team members refine the items they are likely to implement in an upcoming sprint. In SAFe, on none of the levels a backlog refinement event is specified. On the team level, refinement happens continuously within the teams, and having a meeting is optional. On the program and value stream levels, the only constraint for refining is that it has to be well-elaborated before the PI planning. On the portfolio level, epics are refined as long as they go through the Kanban system. Only if they pass, they become part of the portfolio backlog. At this time refinement should already be finished.

In LeSS and Nexus, sprint planning consists of two parts. In the Nexus sprint planning, the product owner and representatives from all teams validate and adapt backlog items, determine the Nexus sprint goal and assign backlog items to the teams. After that each holds its own sprint planning in parallel to plan the assigned items. Similar to that the first part of sprint planning in LeSS is done by product owner and representatives from all teams who focus on shared work and the backlog items that are related. After discussing dependencies they coordinate which team works on which items and with which teams they might have dependencies in the upcoming sprint. The sprint planning two is held in parallel by each team after sprint planning one to discuss design issues and to make a plan for the upcoming sprint which is visualized in Teams' sprint backlogs. Planning in SAFe requires much more effort throughout all levels. Planning is done top-down. Therefore, in the PI Planning, which is conducted for each ART or Value Stream and lasts two days, one needs to take into account the vision and strategic themes defined on the portfolio level as well as inter-team dependencies. With the results in mind, each team conducts its own Iteration Planning.

In each framework, the daily Scrum - or daily Stand-up in SAFe - plays an important role. In Nexus, there is not only a team-internally daily Scrum, but even before that a Nexus daily Scrum with representatives from all teams who discuss integration and dependency issues takes place. After that the teams discuss the identified issues within their teams in the daily Scrum. This allows that integration issues and dependencies are addressed everyday and detected early especially by the affected teams. In LeSS, the daily Scrum takes place team-internally. However, there is the possibility that team members of other teams take part in the daily Scrum of other teams to discuss current dependencies and integration issues. In SAFe, there is a daily Stand-up for each team separately to discuss team-internal issues. There is no focus on integration. The sprint review at the end of each iteration in Nexus and LeSS is an overall event with the customer and different stakeholders to demonstrate the current increment and to incorporate their feedback. While in Nexus the entire new features of the increment are shown at once, the setting of the

LeSS sprint review is meant to be similar to a "Science Fair" with different areas where team members present those new functions their team has developed. In SAFe, the sprint review is divided into Team Demo, System Demo and Solution Demo with different participants, parts of the system shown and different frequency.

Agility not only means to make changes to the developed system continuously but also to the own working frameworks. The sprint retrospective is used to identify areas of improvement. The approach to this event is different for all frameworks. In Nexus, the retrospective has three parts: First, appropriate representatives from all teams identify issues which had impact on more than one Scrum team, making those issues transparent and detecting possibilities to deal with them followed by a team-internal part in which each development team using input from the first part formulates actions to be taken to improve. Finally, representatives from all teams visualize and discuss the identified actions. In LeSS, first a team-internal retrospective takes place to identify areas of improvement for the team members themselves and in the overall process. After that the overall retrospective is held by the product owner, Scrum masters and representatives from all teams to explore possibilities to improve the overall work. SAFe describes the Iteration retrospective held by each team corresponding to Scrum at the end of each Iteration. Additionally, there are Inspect and Adapt workshops at the end of each PI to identify possibilities to improve in the next PI concerning reliability, quality and velocity.

In Nexus, no additional events are specified. However, in practice there probably need to be regular events for the Nexus integration team to discuss integration issues. In LeSS, no additional meetings are defined. Only for LeSS Huge, the described overall events do not take place for the entire product, but for the requirement areas. Those events are not specified in detail, but it is assumed that no additional rules apply. However, there also might be an additional meeting in which the overall product owner and the requirement area product owners can synchronize. In SAFe, there other meetings, such as a product owner sync and Scrum-of-Scrums. As coordination in SAFe, is extremely complex, there might be many more meetings as well.

Summing up, all frameworks use the events defined by Scrum. Though all frameworks have a different approach on how, when and where integration issues are discussed and solved [76, 21, 55, 40, 14, 58].

2 Foundations

		Nexus / Scaled Professional Scrum Schwaber	Large Scale Scrum (LeSS) Larman/Vodde	Scaled Agile Framework (SAFe) Leffingwell
	Sprint	<ul style="list-style-type: none"> - no length defined, Scrum rules apply - all teams start and end Sprints at the same time - all development work is done in the Sprints 	<ul style="list-style-type: none"> - no length defined, Scrum rules apply - all teams start and end Sprints at the same time - all development work is done in the Sprints 	<ul style="list-style-type: none"> - called Iterations instead of Sprints, but are like Sprints in Scrum - two weeks suggested with same start and end time for each team - all development work is done in the Sprints
	Product Backlog Refinement	<ul style="list-style-type: none"> Overall Backlog Refinement - appropriate members from all teams, PO and Scrum Master - first part: decompose and detail items - second part: identify and visualize interdependencies 	<ul style="list-style-type: none"> Overall Backlog Refinement (optional) - PO and representatives from all teams - align the teams - discuss which team could implement which Backlog item 	<ul style="list-style-type: none"> Program and Value Stream Backlog Refinement - is done continuously, no meetings specified - refinement must be well-elaborated before each PI Planning Portfolio Backlog Refinement - is done continuously, no meetings specified - epics haven gone through Kanban system => Refinement done
	Team Level	<ul style="list-style-type: none"> - not defined 	<ul style="list-style-type: none"> Backlog Refinement - held by each team separately - refine the items the team is likely to implement in later Sprint 	<ul style="list-style-type: none"> Team Backlog Refinement - refinement is done continuously, meeting is optional - held by each team separately, discuss and estimate stories
	Sprint Planning	<ul style="list-style-type: none"> Nexus Sprint Planning - appropriate representatives of all Scrum teams and PO - validate and adjust the current status of the Product Backlog - determine the Nexus Sprint Goal - coordinate which items each team works on in the upcoming Sprint 	<ul style="list-style-type: none"> Sprint Planning One - PO and representatives of all teams - discuss shared work and related Backlog items - decide which items will be implemented by which team 	<ul style="list-style-type: none"> Program Increment Planning - lasts two days and takes place every 8 to 12 weeks (PI length) - commit to common goals - conducted for each ART/ Value Stream
	Team Level	<ul style="list-style-type: none"> Team Sprint Planning - held in parallel by each team after Nexus Sprint Planning 	<ul style="list-style-type: none"> Sprint Planning Two - held in parallel by each team after Sprint Planning One - discuss design issues - make a plan for the upcoming Sprint (visualized in Sprint Backlogs) 	<ul style="list-style-type: none"> Iteration Planning - team-internally - define the scope including items and goals for the next iteration
	Daily Scrum	<ul style="list-style-type: none"> Nexus Daily Scrum - appropriate representatives of each Scrum team - detect and discuss integration issues as well as dependencies 	-	-
	Team Level	<ul style="list-style-type: none"> Daily Scrum - held by each team separately - discuss integration issues from Nexus Daily Scrum affecting them 	<ul style="list-style-type: none"> Daily Scrum - held by each team separately - members from other teams taking part to share information 	<ul style="list-style-type: none"> Daily Stand-up - held by each team separately
	Sprint Review	<ul style="list-style-type: none"> - overall review of the entire increment - like the corresponding Scrum event - incorporate feedback of customers and other stakeholders 	<ul style="list-style-type: none"> - overall review of the entire increment - like a "science fair" with different areas where team members present those new functions their team has developed - incorporate feedback of customers and other stakeholders 	<ul style="list-style-type: none"> Team Demo - with agile teams, Business Owner and other ART stakeholders - review the increment after each iteration - incorporate feedback System Demo - at the end of the iteration - aggregated view of functions implemented by all team in the ART Solution Demo - progress of a value stream is demonstrated at the end of PI
	Sprint Retrospective	<ul style="list-style-type: none"> Nexus Sprint Retrospective (Three parts) 1. Part: appropriate representatives from all teams identify issues which had impact on more than one Scrum team, making those issues transparent and detecting possibilities to deal with them ... 3. Part: representatives from all teams visualize and discuss the identified actions 	<ul style="list-style-type: none"> Overall Retrospective - takes place after team Retrospective - PO, the Scrum Masters and representatives from all teams - explore possibilities to improve the overall work 	<ul style="list-style-type: none"> Inspect & Adapt Workshop - at the end of each PI - identify possibilities to improve in the next PI concerning reliability, quality and velocity
	Team Level	<ul style="list-style-type: none"> 2. Part: each Development team using input from the first part formulates actions to be taken to improve 	<ul style="list-style-type: none"> Team Retrospective - corresponding to the Scrum Retrospective - held by each team - takes place before the overall Retrospective 	<ul style="list-style-type: none"> Iteration Retro - each team discusses the last iteration and possibilities to improve
	Further Events		<ul style="list-style-type: none"> LeSS Huge: Area Sprint Planning, Backlog Refinement, Retro, Review - take place for each Area instead of for the whole product - no specified additional rules for those events in LeSS Huge 	<ul style="list-style-type: none"> Scrum-of-Scrums - Release Train Engineer and Scrum Masters - taking place weekly or even more often - make progress and any impediments transparent PO Sync - Product Management and POs - discuss progress of the ART concerning the PI objectives

Table 2.6: Process and Events in Nexus, LeSS and SAFe (see Sections 2.3, 2.4, 2.5 for descriptions and sources)

2.6.4 Artifacts

The central artifacts are represented by Table 2.7 and explained in the following.

	Nexus / Scaled Professional Scrum Schwaber	Large Scale Scrum (LeSS) Larman/Vodde	Scaled Agile Framework (SAFe) Leffingwell
Product Backlog	<ul style="list-style-type: none"> - one for all teams - maintained by PO 	<ul style="list-style-type: none"> Product Backlog - one for all teams - maintained by PO LeSS Huge: Requirement Area Backlog - all items the teams in the Requirement Area work - owned by Area Product Owner - replaces Product Backlog in LeSS Huge 	<ul style="list-style-type: none"> Team Backlog - if Scrum is used at the Team Level, same as for Scrum - contains user and enabler stories Program Backlog - includes the upcoming features that affect the solution Value Stream Backlog - includes the upcoming capabilities that affect the solution Portfolio Backlog - includes Business and Enabler Epics which passed the Kanban System
Sprint Backlog	<ul style="list-style-type: none"> Nexus Sprint Backlog - composite of the items in the Team Sprint Backlogs - makes dependencies transparent Team Sprint Backlog - items the team works on in the current Sprint 	<ul style="list-style-type: none"> - not specified, items from the current Sprint are specified as such in the Product Backlog - separate for each team - items the team works on in the current Sprint 	<ul style="list-style-type: none"> not specified for Program, Value Stream, Portfolio Backlogs Team Level - if Scrum is used at the Team Level, same as for Scrum
Sprint Goal	<ul style="list-style-type: none"> - overall Nexus Sprint Goal for all teams in the sprint 	<ul style="list-style-type: none"> - overall Sprint goal for all teams in the sprint 	<ul style="list-style-type: none"> - Iteration goal for each team - PI goals on the program/ value stream level - Portfolio level follows strategic themes
Product Increment	<ul style="list-style-type: none"> - Integration is done by Nexus Integration Team - usable and potentially releasable at the end of each Sprint 	<ul style="list-style-type: none"> - integrated and potentially shippable at the end of each Sprint - integration is responsibility of each team and done in the Sprint 	<ul style="list-style-type: none"> Program Increment (PI) - integrated Increment built by an ART
Definition of "Done"	<ul style="list-style-type: none"> - all teams know it and adhere to it - Integration Team evolves it 	<ul style="list-style-type: none"> - all teams know it and adhere to it - agreed on in the beginning but evolves over time 	<ul style="list-style-type: none"> - Different for Increments on each level and for releases
Additional Artifacts	<ul style="list-style-type: none"> User Stories - to be implemented by the teams 	<ul style="list-style-type: none"> User Stories - to be implemented by the teams 	<ul style="list-style-type: none"> User Stories (Team Level) - to be implemented by the teams Features (Program Level) - describe system behavior and are split into stories to be implemented Capabilities (Value Stream Level) - describe solution behavior and are split into features to be implemented Epics (Portfolio Level) - Business Epics deliver value to the customer directly - Enabler Epics evolve the architectural runway ... Vision (future objectives of the solution), Roadmap (makes Value Stream and Program Level Deliverables transparent including a timeline), Milestones (progress points, e.g. PI milestones)...

Table 2.7: Artifacts in Nexus, LeSS and SAFe (see Sections 2.3, 2.4, 2.5 for descriptions and sources)

Especially in Nexus and LeSS, the names of the artifacts are mostly the same as defined by Scrum. In SAFe, which adds further artifacts, some of the names of the artifacts are adapted, such as system increment instead of product increment and team or program

backlog instead of product backlog [32].

While in Nexus and LeSS the central artifact is a single product backlog which is maintained by a single product owner, in SAFe every team has an own product backlog maintained by the according product owner. Additionally, in SAFe, backlogs exist on all levels including features on the program level, capabilities on the value stream level and epics on the portfolio level.

In all frameworks, there are sprint backlogs including all backlog items of the current sprint. In Nexus, there are not only Team sprint backlogs but also an Nexus sprint backlog which is the composite of all Team sprint backlogs. LeSS refers only to Team sprint backlogs, but with an existing overall product backlog it might be easy to filter accordingly to see an overall sprint backlog.

Additionally, in both Nexus and LeSS, there is an overall sprint goal all teams know and each team has to deliver its part at the end of the sprint to reach the goal. In SAFe, the team sprint backlogs exist as well, while on the other levels sprint backlogs are not specified, but are probably part of the according backlogs on the levels. Each team in SAFe defines an iteration goal. The equivalent to this on the other levels are PI goals on the program/value stream level and strategic themes on the portfolio level.

In all frameworks, the teams work on overall integrated product increment. Nexus and SAFe require a usable and potentially releasable product at the end of each sprint, while in SAFe releases should be done with a fixed frequency, however not every Iteration. A potential release cycle length could be the length of a PI.

All frameworks include that there should be a definition of "done", though every project team in an organization has to define and evolve it itself. In Nexus, the integration team is responsible for defining what "done" means in the beginning and to adapt it over time. In LeSS, the teams agree on a suitable definition of "done" in the beginning, but evolve it over time. In SAFe, the definition of "done" differs on all levels and potentially also between the teams.

There might be another definition of "done" for the releases. Additional artifacts are user stories in all frameworks. While Nexus and LeSS do not require further artifacts, in SAFe next to user stories, there are features, capabilities and epics as well as strategic constructs, such as vision, roadmap and milestones.

To sum up, Nexus and LeSS rely on central used artifacts, such as the product backlog and the sprint goal, to keep the whole product focus. Opposing to that in SAFe all team level artifacts are only related to one team, while the coordination between the teams is done on the program level [76, 21, 55, 40, 58].

2.6.5 Architecture

An overview of the role of architecture in the frameworks is given in Table 2.8 followed by an explanation in this section. When it comes to Nexus and architecture there is no information to be found. The only thing that is mentioned in the Nexus Guide is that the Nexus integration team knows the architectural standards of the organization and teaches

the development teams to adhere to them. There is no guidance on how this should be done in practice. Therefore, one can assume that - as in Scrum - Schwaber considers the architecture to evolve, but that there has to be some overall guidance [76]. The other two frameworks see architecture as a much more essential topic and therefore address it comprehensively. LeSS emphasizes emergent design and defines the code itself as the real architecture of a system. Based on this assumption architecture grows along with programming. To achieve a high-quality design the focus in LeSS is mostly on behavior oriented tips. Further, it is assumed that organizational structure influences culture. A culture of continuous improvement, learning and coaching is required. On the other hand, SAFe emphasizes that emergent design and intentional architecture coexist and contribute to each other fostering the architectural runway of the system.

In LeSS, the role of an IT architect within the team is not explicitly defined. All developers are architects in LeSS. However, some team members are more experienced in architecture and therefore are able to coach the others or to conduct architectural spikes. In SAFe, on the team level no architecture role is defined as it is assumed that architecture emerges. On the program and value stream level the role of system and solution architects exist who can be either one person or a team of architects. They provide guidance to the teams or ARTs respectively. At the portfolio level, additionally the role of the enterprise architect exists who has an overview of all levels, provides strategic architectural guidance and owns architectural enabler epics. The enterprise architect mostly focuses on intentional architecture. The architects on the lower levels split up architectural epics into capabilities and features for incremental implementation. In LeSS, the role of an enterprise architect is not defined. Larman and Vodde argue that only architects that are in touch with the code are required and no "Powerpoint" or "Ivory Tower" architects [20].

In SAFe, architecture is addressed in the process by incrementally building the architectural runway, implementing (architectural) enabler epics now to support features in the future, conducting Spikes and establishing CoPs. Additionally, an advise is to keep the architecture as simple as possible and that the ones who build the software test it. In LeSS, hands-on master-programmer architects in the teams provide some guidance. A culture of excellence in code by coaching needs to be established. Practices, such as pair-programming, test-driven development, refactoring, conducting Spikes and establishing CoPs are used. Larman and Vodde also recommend to establish a tiger team in the beginning of a project to explore architecturally significant issues and to spread their knowledge in the teams later e.g. in architecture documentation workshops. In SAFe, no events for exploring architecture are defined. In LeSS, it is suggested to hold agile modeling and design workshops in which feature teams model related to their upcoming objectives or even to overall system architecture by using various agile modeling techniques on whiteboards whenever modeling is useful. Further, teams are encouraged to use modeling throughout the process also outside of workshops. Modeling throughout the development process is seen as a helpful tool whatever method is used. Larman and Vodde leave it to the teams to decide which modeling method helps the most. They also encourage the teams to document models visible for the entire team on whiteboards in the team room and in a shared

	Nexus / Scaled Professional Scrum Schwaber	Large Scale Scrum (LeSS) Larman/Vodde	Scaled Agile Framework (SAFe) Leffingwell
Emergent Design vs. Intentional Architecture	- assumption based on Scrum: purely emerging architecture	- emphasis on emerging architecture - architecture is grown not built - source code is the true architecture - certain behaviors and culture drive high-quality design	- emerging architecture on the team level - intentional architecture above team level - all levels contribute to each other and the architectural runway
Role of IT Architects	- not defined	- as the source code is assumed to be the architecture, everyone who changes the code, changes the architecture - as teams are cross-functional, it make sense that one or more team members is an expert on architecture	System Architect/ Engineer at Program Level - one person or a team - define the overall architecture of the large-scale system, non-functional requirements and interfaces Solution Architect/ Engineer at Value Stream Level (optional) - one person or a team - specifies overarching architecture to connect the solution across ARTs - guide the System Architects/Engineers in the ART
Role of Enterprise Architects	- not defined - but intgration team should know architectural standards of the organization and teach the teams	- not defined - no "Ivory tower" or "Powerpoint architects" wanted	- role of the enterprise architect is defined on the Portfolio Level - has an overview of all levels - provides strategic architectural guidance - defines and owns Enabler Epics
Architecture in the Development Process	- not defined	- hands-on master-programmer architects - culture of excellence in code - pair-programming - coaching for high-quality code and design - test-driven development and refactoring - conducting Spikes - Communities of Practice (CoP) - establishing a tiger team in the beginning	- Building the architectural runway - implementing (architectural) Enabler Epics now to support features in the future - conducting Spikes - Communities of Practice (CoP) - keep the architecture as simple as possible - the one who build the software test it
Events for Designing/ Architecting	- not defined	Architecture Documentation Workshops - at the end of a tiger team phase Agile Modeling and Design Workshop - feature teams model related to their upcoming objectives or even to overall system architecture - agile modeling techniques ("vast whiteboards") - before building a new Backlog item or just-in-time whenever useful - Participants depend on the subject that is modeled	- not defined
Tooling/ Modeling	- not defined	- whiteboard modeling with in whatever form required - not only in design workshops, but also in daily work - document models on whiteboards in the team room and in a shared Wiki Space - models evolve over time	- "When in doubt, code or model it out" - "If you model one thing in agile, model the domain"
Recommended Practices	-not publicly available	-Domain-driven Design	-Domain-driven Design

Table 2.8: Architecture in Nexus, LeSS and SAFe (see Sections 2.3, 2.4, 2.5 for descriptions and sources)

wiki space. Models are said to evolve over time [20]. In SAFe, domain modeling is encouraged. The role of modeling in SAFe is best described by two citations: "When in doubt, code or model it out" and "If you model one thing in agile, model the domain" [41]. In both SAFe and LeSS DDD is recommended. How DDD can support architecture scaled agile organizations is explained in Section 3.2.1 [20, 41, 40].

2.7 Domain-driven Design (DDD)

In the next sections, the major ideas of DDD are explained. Both strategic and tactical DDD concepts are presented.

2.7.1 Approach and Objectives

DDD is an approach to the design and development of complex and large-scale software systems [31]. Initially introduced in 2003 and coined by Eric Evans, the premises of DDD are twofold. First, the focus in most software projects should be on the domain and its business logic. Second, complex domain designs need to be based on a model of the domain [30]. Therefore, DDD aims to provide a way of thinking and a set of priorities to handle the complexities resulting from the business activities and logic in the domain to accelerate development [88].

A domain in DDD is defined as "a sphere of knowledge, influence or activity" [31]. A major goal of DDD is to make this domain knowledge explicit through a model and an ubiquitous language, helping a team to influence the domain by acting according to the built up knowledge. Further Evans (2003) motivates the approach by the fact that while some complexities arise from the technical design of an application, the most significant complexities result from the business logic itself. There are many approaches to deal with technical complexities and associated design issues, such as design of networks and databases, but one also has to deal with the complexities of the business as a project will not benefit from a well conceived technical infrastructure as long as this central aspect is not well handled [30]. In the following the central principles of DDD are presented.

Focus on the core domain. In DDD one of the most essential principles is the focus on the core domain which is the part of the product most crucial for providing business value and generating competitive advantage [64]. From a strategic point of view an organization has to excel in its core domain. A project concerning the core domain has the highest priority and is supposed to get the best qualified team [83].

Bounded context. The Bounded context explicitly sets boundaries to a model concerning the model itself, as well as team organization and all implementations. The bounded context is the part of the business in which a model applies. In each bounded context one team is at work that continuously works on code and the model within the context [31]. To get a deeper insight on how to define bounded contexts see Section 2.7.4.

Creative collaboration. Models of the domain are explored by developers and business experts in creative collaboration [31]. DDD aims to gain deep insight into the problem domain and its key concepts by creating a model reflecting a high amount of knowledge

about the domain. This can only be reached by a close and continuous collaboration between domain experts who have the business knowledge and developers who know how to build software [30].

Evolving the model constantly. The team has to reconsider constantly if the current model helps to solve the business problem in focus. Especially, new business cases may require the model to change to make the underlying concepts more explicit. Therefore, constant knowledge crunching to maintain, simplify and enhance the model are crucial. Exploration and experimentation help to find the most useful design for the model and the associated code. Not the first usable design should be chosen. It is suggested that for each useful design at least three bad ones exist. This will lead to deeper insights in the domain and helps the team to understand the business better [64].

Continuous Integration. The more people work on a business problem the more likely code and model are to fragment. To avoid fragmentation a process to merge all implementation artifacts frequently needs to be established. Model and code are closely linked and therefore a refactoring in the code means also making changes to the model. Inversely changes in the model affect the code [31].

Ubiquitous Language. To allow effective communication between domain experts and developers a common language needs to be defined. This ubiquitous language is essential to the success of DDD as communication without a language which is shared by developers as well as domain experts cannot be effective. The language is developed alongside the model and the code as the same terms should be used in model, code and language. Changing the model or the code, means changing the language [64]. The term ubiquitous language and how such a language can be established and evolved in the DDD process is further described in Section 2.7.3.

DDD helps with the following major aspects. It brings a team of domain experts and developers together to be able to create software reflecting an explicit model of the business which was before only present in a mental model of the domain experts. This approach unifies domain experts and software developers while forming a team with deep knowledge of the related business domain and an ubiquitous language as the central tool to communicate and work together effectively toward a common goal [83].

DDD not only provides guidance on team level but also on a strategic level, by addressing strategic initiatives of the organization. Therefore, one can distinguish between strategic level (see Section 2.7.4) and tactical level DDD (see Section 2.7.5). The central concept of strategic design is context mapping, while the DDD building blocks are essential on the tactical level. Figure 2.6 gives an comprehensive overview of these concepts. Strategic design follows the strategy of the company. It defines the relationships between bounded contexts and the related teams giving possibilities to detect potential project or software failures caused by interdependency between the contexts [83]. The most crucial strategic



Figure 2.6: DDD Building Blocks and Context Mapping [31]

design tools of DDD are bounded context, distillation, context mapping and large-scale structure [30]. Additionally, tactical design helps the teams to deliver software meeting the specifications of the domain experts while also meeting technical demands. Tacti-

cal design modeling tools allow to develop software which is scalable, testable and of high quality [83]. Tactical design especially concerns the intra-team collaboration within bounded contexts, while strategic design helps to understand the relationships between bounded contexts and gives an overall understanding of the entire business domain. The main focus in this thesis lies on strategic design. Before elaborating strategic design the role of the domain model in DDD is explained and the most crucial principles of DDD, bounded contexts and Ubiquitous Language, are addressed. The most essential part of Tactical DDD, such as the domain model and its building blocks, are explained shortly in Section 2.7.5.

2.7.2 Domains, Subdomains and Bounded Contexts

A domain is what a business does in its surrounding environment.

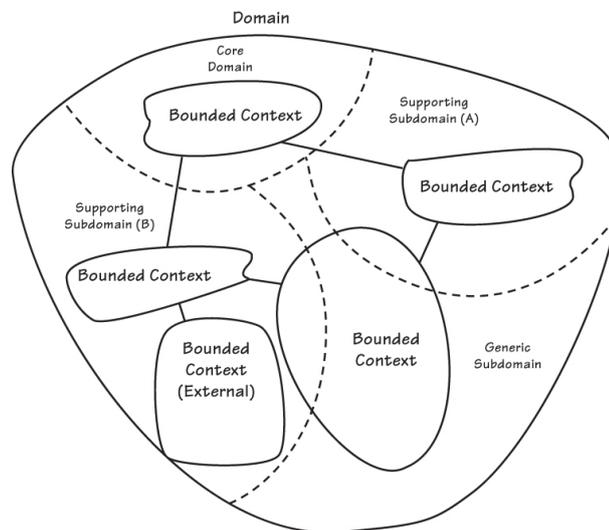


Figure 2.7: Domains, Subdomains and Bounded Contexts [83]

Each organization establishes know-how and a set of methods to sell certain kinds of products or services in certain markets. All this is part of the domain of the organization. If a team develops applications for an organization, they work in the organization's domain. According to this very broad definition, the domain in which an organization acts in is quite obvious. This notion of the term domain can lead to the misunderstanding that DDD means to define a single, large and extremely complex model of an organization including all aspects of its business domain. However, DDD wants to reach exactly the opposite assuming that the whole business domain can be divided into subdomains. Each subdomain addresses a certain area of the business supporting teams to focus on a certain area for which it should solve a certain problem [83]. For each subdomain, a logical model

should be defined to gain deep understanding of this distinct area of the business [64].

In the best case, one bounded context is at work in only a single subdomain. However, it is fine if there are more than one bounded contexts in a subdomain. If a subdomain was developed utilizing DDD from the beginning, each subdomain would correspond to one bounded context worked on by one team, but this is especially in large and established companies seldom the case [84].

In the example in Figure 2.7 all bounded contexts except the one on the bottom which overlaps different subdomains are good examples for how subdomains and bounded contexts relate. However, in the best case, there is a one to one alignment between a subdomain and a bounded context.

2.7.3 Establishing an Ubiquitous Language

Domain experts use their own language to describe terms within the domain which is difficult to understand for the software developers. Software developers might understand this description of the desired functionalities in a different way. On the other hand, software developers describe the technical terms of the software to the domain experts in their own language which is not understandable to domain experts. This may happen if one part of the team has to translate e.g. technical terminology into terminology of the domain or the other way around as this leads to a loss of information. This bears the risk that even if domain experts and developers communicate frequently about the specifications of the software, they might mean a complete different thing. The resulting software will not fulfill the specifications which are required to provide business value to the domain. To avoid these misunderstandings, DDD suggests to establish an ubiquitous language which is shared by domain experts and developers. The shared language should be developed in collaboration of all team members and needs to be exercised relentlessly not only in oral and written communication, but also in all documentation and code. Therefore, the language and the model need to be closely linked and evolve alongside each other [30]. As any language the ubiquitous language of a bounded contexts evolves over time. Changing terminology of the language means a change to the model as well as the code [84].

When starting to establish an ubiquitous language one should not only use the language of domain experts or developers, but use parts of both, and additionally it might be necessary to create new terminology [31]. It is important that everyone in the team consistently uses a clearly defined word for a specific concept to avoid confusion. The terminology should be unambiguous meaning that no overloaded terms or terms which have a very specific meaning in software development should be part of the Ubiquitous language. The common language is visible in the code by conforming class, method and property names to the ubiquitous language. Finally, it can help to define a glossary with crucial terminology for the team [64].

2.7.4 Strategic Design

As most organizations are too large and far too complex to be represented by a single comprehensive model, they have to be broken down into several smaller models. This has to be done for concepts and implementation. However, the different models need to be able to interact with each other. To integrate the various parts of the business domain strategic design provides a set of themes, such as bounded contexts, context mapping, distillation and large-scale structure [30]. Strategic design focuses on large models divided into several bounded contexts as well as team organization [34].

Defining Bounded Contexts Bounded contexts are the most central topic of DDD. As one large unified model can lead to misunderstandings, as different groups of people communicate using different vocabulary. The same term could mean a completely different thing which could lead to a lot of confusion [34]. Therefore, in large organizations multiple models should co-exist which apply in different contexts. Often their existence is not recognized until code from different models is combined in one software or two interacting softwares. Then the code becomes difficult to understand, more prone to errors and unreliable [30]. It might be clear that another software's data format and technical specifications differ, but there can be more crucial differences if each software has an own underlying model which is only implicit in code without being formulated explicitly in the code or in models. Differences are even more unlikely to be detected if they lie in the same code base. According to Evans (2003) this happens in all big projects. Problems will be recognized in the end if the code does not work correctly. The problems start long before that in the way teams are structured and interact. As a result, to define clear bounded contexts one has to look at both team organization and (end) products [30]. Other factors for determining boundaries of contexts are human culture and especially the language a certain group of people uses [34].

A bounded context in DDD is defined as "a description of a boundary (typically a subsystem, or the work of a particular team) within which a particular model is defined and applicable" [31]. The bounded context clarifies what a model is used for, in which area it should be consistent and what is out of scope. By making the boundaries explicit DDD ensures that the focus lies on the problem within the bounded context which needs to be solved without being distracted by issues outside the context [64]. Often bounded contexts are compared to language or country boundaries [84, 64]. Within each of the bounded contexts a different ubiquitous language should be defined and exercised. Then everyone in the team can communicate with each other without misunderstandings which can happen during translations. If people are talking to others outside their bounded context, the language can be completely different [84]. Millet (2015), compares the boundaries of a context to border control meaning that nothing should pass the borders without being checked and valid. Everything that passes should adhere to the defined rules and use the defined language [64].

Defining boundaries for a context is challenging and the bounded context evolves as the

teams learn more about the domain. One of the key goals in strategic design is to align bounded contexts to Subdomains. Naturally, also team organization can be an indication for defining context boundaries. In the best case, one team would work on one Bounded Context which corresponds to one subdomain [84]. This would mean that a bounded context aligns with a certain area of the business. Looking at Figure 2.8 this would apply for subdomain A.2 and A.3 with their corresponding bounded contexts A.2.1 and A.3.1. A bounded context includes a “vertical slice of functionality from the presentation layer, through the domain logic layer, on to the persistence, and even to the data storage” [64]. Therefore, each context is responsible for its presentation, domain logic and persistence. This allows the team in the context to direct its work towards delivering business value with the most suitable implementation on each layer for its own context. The domain logic is represented as the domain model which is implemented in the software. Therefore, Figure 2.8 makes clear that each bounded context has its own model. This means that objects might exist in different contexts but most likely have different attributes and logic. This could be the case for the entity product in the contexts A.2.1 and A.3.1. The product can have completely different attributes and logic in context A.2.1 and in A.3.1.

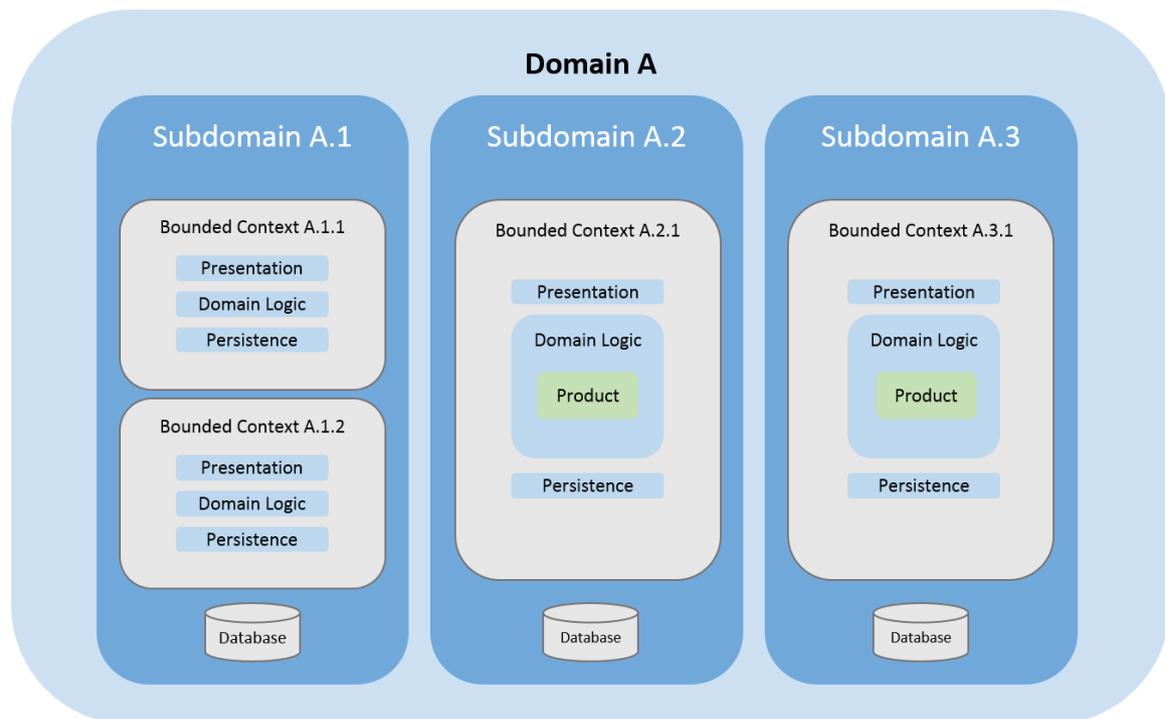


Figure 2.8: Vertical Subdomains and Bounded Contexts with DDD [64]

Isolating the subdomains prevents undesired side effects that can occur if changes are made to one of them [64]. However, not all contexts can be completely isolated, but they

relate to each other. In DDD, these relationships are made explicit through the use of context maps.

Context Mapping Each bounded context in DDD should be able to evolve as independent as possible from other contexts. However, all bounded contexts are related to other contexts. A context map represents the integration between bounded contexts in a simple and graphic way. It shows the current and real relationships between the contexts rather than the desired state. However, the context map should be adapted according to the changes made to the system [84]. A first context map helps to get a better understanding of how the contexts interact. Further, a context map can give an overview of the architecture of systems including not only the relationships between the contexts but also to get a comprehensive overview of which domain model is used in the different contexts [89]. However, not only architectural aspects, but also organizational and political aspects should be taken into account for a context map. Team organization and communication are one of the most important clues for defining a relationship between bounded contexts. If two teams need to communicate a lot during development, their models and context might be more closely related and dependent on each other. Even the actual office space can have an impact. Teams that work in different locations or different parts of the same building communicate less and therefore their model will diverge more likely. Further, isolation and sharing mechanisms as well as influence on each other should be taken into account [31]. Evans highlights that having a clear view on the links between team organization, software models and design is essential to the success of software projects. Therefore, this information should also be part of the context map [30]. Even if a context map is not an enterprise architecture diagram, it can be the basis for higher level architectural decision or investigations providing the explained further information. Context maps may help to identify integration bottlenecks, issues with governance and other architectural deficiencies [83].

DDD defines different kinds of relationships between bounded contexts: partnership, shared kernel, customer/supplier, conformist, anticorruption layer, open-host service, published language, separate ways and big ball of mud [83, 30]. These relationships exist between two teams of which each is responsible for one bounded context.

In a partnership the two teams have as dependent set of goals. They either both fail or succeed. The two teams are required to meet frequently to synchronize their work as well as their timelines. Further, they should use continuous integration to achieve consistency of their work. A high level of commitment is required. Therefore, the relationship between the contexts should be only maintained as a partnership if it really is advantageous for both teams. The advantages need to balance the commitment and the integration efforts. Over the long term it often makes sense to reduce commitment and find another way to integrate [84].

Shared kernel describes a relationship in which two teams that are working on the same application on bounded contexts that are overlapping. Therefore, the two teams share a

part of the domain model as well as of the code. The part of the model that is the same for both teams is called the shared kernel [64]. Both teams need to know which objects in the model they share and teams often need to communicate and agree on change to the shared elements. It is very challenging to determine which part of the model to share and to maintain it in a suitable way for both teams [84].

In customer/supplier relationship there are two teams that have a upstream (supplier) to downstream (consumer) relationship. This means that the upstream team is to provide functionality according to what the customer needs. The downstream team needs to request domain model changes at the upstream teams. There have to be planning sessions in which the tasks required by the downstream team are negotiated and budgeted. This potentially slows down both teams. The upstream team needs to fulfill the downstream team's tasks and the downstream team also has veto power if the upstream team wants to make changes to the model which are not suitable for them. However, the downstream team really depends on the upstream team and sometimes can only wait for the requested changes to be implemented [30].

A conformist relationship exists between an upstream and a downstream team if the supplier (upstream) is not motivated to implement the changes the downstream team requires. The downstream team has to conform to the upstream team's model if the team can't make the effort to translate the upstream model. Becoming a conformist can happen especially to small and newly established teams when they integrate with an upstream team with a very complex, large and well-established model [84]. It is also common if a team needs to integrate with an external supplier. Here it might be useful for the downstream team to create an anticorruption layer to retain integrity and clarity of their model and changes on the upstream model do not affect their model [64].

An anticorruption layer is an isolating layer which a downstream team can use to provide itself with functionality from the upstream teams in a way that corresponds to its own model. The layer translate in both directions between both models [30]. Downstream teams should create an anticorruption layer to protect their own models so that they can adapt their model according to their own business needs [84].

An open-host service is a protocol a team can create to provide access to its functionality to other teams via services. The protocol should be available to all teams so that they are able to integrate. If various other teams require changes to the protocol, the providing team should adapt it. However, if only one team requires changes for a very special use case, the team should not adapt the protocol but provide a one-off translator to the other team to amend the protocol for that use case [30].

A published language constitutes a shared language which is well-documented and allows exchange of information. This language allows that other bounded contexts can consume and translate domain information easily. Defining a published language can be achieved w.g. with XML or JSON schema. Often open-host services and published language can be connected to provide a better integration to other contexts [84].

Separate ways means that a bounded contexts has no relationship to any other context allowing the team to find simple solutions that are targeted only on its own business needs

[30].

A big ball of mud often exists in large and established systems. It describes parts of the system with mixed models and undefined boundaries. It is suggested to draw boundaries around these parts and declare them as big balls of mud trying to avoid that the models corrupt other models in other contexts. It might be too late to apply sophisticated modeling in such a context [31].

Distillation Another concept included in DDD is distillation. The goal of distillation is to find out which problem area is the most important one in the developed software [50]. As in a large project not all parts of the design can be refined in the same way, one has to set priorities to determine the subdomains which are most crucial to the success of the business [30]. One can distinguish between core, supporting and generic subdomains. The core domain is most crucial to the success of the business. The teams should figure out what actually belongs to the core and what not. Moving certain function out of the core domain allows a better focus on what really is essential [50]. A project should invest in a well-defined model and ubiquitous language for the core domain. Distinguishing from competitors in the core domain gives an organization a competitive advantage. The best developers and domain experts should work on the core domain to develop a supple design [30]. For supporting subdomains an organization does not make such high investments as for the core subdomains. Software in the supporting subdomain is necessary because the core subdomains couldn't be succeed without it. Custom development is required in the supporting subdomain but one could contemplate outsourcing. In the generic subdomains off-the-shelf solutions can be used or their development can be outsourced. Organizations shouldn't make high invests in the generic subdomains. Instead of developing applications themselves, buying standard software might make more sense [84].

Large-Scale Structure Large-Scale Structure becomes important as soon as a context map itself grows very large and complex. The context map can be overwhelming and might become unmanageable [50]. A large-Scale Structure is a language giving an overview of the entire system on whose basis the overall structure can be discussed. Therefore, a set of rules, roles and relationships is needed which that support design as well as understanding of the system. It especially helps to understand the role of single parts in the system as a whole without knowing detailed what the part's responsibility is [31]. System Metaphor, Responsibility Layer and Evolving Order are the most known approaches. System Metaphor supports the definition of a fundamental structure for the system as a whole. Responsibility Layer implies that the system is divided into several layers, but not in a technical way (UI, logic and database). Each layer is able to call the layer below but not the other way around. Evolving order is not pre-definition of the overall structure of a system in the beginning by a certain logic, but means that the structure will evolve over time depending on the individual components [89].

2.7.5 Tactical Design

On the tactical level, the most essential concept is the domain model itself consisting of certain building blocks: entities, value objects, events, services, modules, aggregates, repositories and factories [31].

Entities are domain objects that are clearly defined by their identity rather than by their attributes. Each entity can be distinguished from any other entity using this identifier regardless of its attributes or history. Entities should be kept spare adding only the most essential behaviors and attributes to it. The definition of an entity should be focused on life cycle continuity [30].

Value objects are defined by their attributes, logic and functionality rather than an unambiguous identity are value objects. These should be treated as immutable [31].

Domain events reflect something that has happened in the domain. Only those events that are relevant because they influence other objects should be modeled. These discrete events can be used as a form to monitor changes or as a means of communication between aggregates [64]. Domain events are different from system events that represent activity in the software. However, domain and system events can be associated, e.g. as a response to each other [31].

Services are activities that do not belong directly to an entity or a value object [30].

Modules are parts of the domain model with high cohesion within them and low coupling between them. Modules make it much easier to understand the underlying concepts of the model and the code. If a person wants to get an overview, he/she can look at the modules and their relationships without being overwhelmed by many details within them. The other way around one can look detailed at one module without be distracted by the other elements [30].

Aggregates are composed of various domain objects, such as entities and value objects [89]. The boundaries of an aggregate should be clearly defined and one of the objects within the Aggregate is the root. Objects from outside the aggregate can only access objects within the boundary through the root. Invariants that apply to all objects inside the Aggregate can be enforced as a whole [30].

Repositories query access to entities and aggregates. While the domain logic should be kept in the model, accessing and storing objects is delegated to Repositories [31].

Factories generate instances of complex domain objects. Factories serve for encapsulation and do not need to have further responsibility in the domain model [30].

The building blocks are used to represent the logic and behavior in the domain. The building blocks as components of the domain model constitute an Ubiquitous Language, on which not only communication, but also code is based [64].

2.8 Related Work

In the following, papers addressing the collaboration between architects and their stakeholders, agile scaling, DDD and the role of architecture in agile environment are summed up.

2.8.1 Bente et al. (2012)

Bente et al. (2012) address next to classical enterprise architecture management (EAM) approaches lean and agile EAM. In this context, they define three major building blocks of lean and agile EAM: "Get Rid of Waste by Streamlining Architecture Processes", "Involve all Stakeholders by Interlocking Architecture Scrums" and "Practice Iterative Architecture through EA Kanban" [13]. Removing waste in this case means to reduce overly strong governance and ineffectiveness in the process of the EA group. Introducing architecture Scrums aims at adjusting the perspective of the EA organization to foster cooperation between architects and their stakeholders and to overcome silo thinking. Getting input from stakeholders helps architects to get a more realistic view on the consequences of their governance activities. Using a Kanban board and an agile backlog for lightweight planning helps to increase transformation speed. Further, they stress participation instead of relying on experts enforcing top-down governance. In the work by Bente et al. (2012), the focus is on collaboration between enterprise architects and their stakeholders to be able to profit from the diversity of knowledge in the company as an input for architectural decisions [13].

2.8.2 Keller (2017)

Keller (2017) focuses on the future of EAM impacted by digitization [47]. He describes the concepts of agile and lean EAM as well as the concept of agile architecture, as also addressed by SAFe. Agile architecture is often connected to REST services, microservices or self-contained systems, cloud and DevOps. Especially microservices or self-contained systems can - in his opinion - only work if the business architecture is defined by business domains divided by clear boundaries. A good model of business capabilities helps defining these boundaries. Therefore, he also names the concept of DDD as a support for determining bounded contexts. Keller (2017) argues that not only the processes, but also the software itself must be agile to be able to react quick and flexible to changing requirements. However, he addresses that some concepts, such as microservices or self-contained systems, were not tested enough so far. He especially addresses that problems might occur if a very high number of developers is working on a large number of microservices. Essential questions according to Keller (2017) are, how to avoid too much redundancy and how to do governance in this environment [47].

2.8.3 Dikert et al. (2016)

Dikert et al. (2016) conduct a case study with focus on challenges and success factors of the adoption of lean and agile methodologies at large scale. Among the challenges are general resistance against change, coordination in multi-team settings, low acceptance of top-down management decisions, difficulties to implement agile methodologies and integration of non-development functions. Success factors include management support, commitment to change, mindset changes, finding a suitable customized agile approach, engaging people, training as well as communication and transparency [27]. Keeping the success factors in mind a customized agile approach is necessary which engages people and leaves time for learning for all involved roles. Further, support from all stakeholders including management is necessary.

2.8.4 Rost et al. (2015)

Rost et al. (2015) focus on how architecture can be integrated in agile development. They explain possible architectural activities, responsibilities and how to document architecture in agile development. Their recommendations include to check compliance to architectural decisions in code reviews, not to introduce many additional meetings, to make clear that documentation and agile are no contradictions. They stress that in multi-team projects the teams need to align especially when it comes to architecture and that the business value of architectural activities needs to be conveyed to the teams.

3 Framework for (Enterprise) Architecture in Agile Teams

The following chapter includes the case study starting with a description of the current situation and the major challenges arising from it. As an approach to solve the problems which occurred, the idea is to bring concepts from agile scaling as well as DDD together. This approach means to support the agile teams by defining cross-team coordination as well as by addressing architectural questions. Finally, a framework for (enterprise) architecture in agile teams is defined. The framework includes roles, a process, artifacts and tools to support cross-team coordination and to establish strategic and tactical DDD in the agile teams. DDD especially is used to define the architecture. The case study itself was done by enterprise architects and is described from their point of view.

3.1 Case Study in an international Insurance Enterprise

This section describes the current situation of the agile teams which take part in the case study. It gives a general overview of the position of the agile teams in the enterprise as well as about their roles, process, artifacts and tools. Finally, the major challenges are described.

3.1.1 Overview

Facing the challenges of digitization, the large insurance enterprise decided to establish agile teams in 2016. These few agile teams coexist next to many other teams that still use waterfall methodologies. Employees from different parts of the enterprise - from the IT department as well as business experts - come together to form cross-functional agile teams. These agile teams are co-located at another location of the company to make sure they can focus on the product, that they develop, without being distracted by their former responsibilities in the traditional part of the enterprise. As agile methodologies were not commonly used in the company before, the new teams received training concerning agile methodologies before they got started. They further obtain training and guidance during the development process.

In this case study the focus is on three of the agile teams which belong along with a few other teams to one major project. The overall project's aim is to develop an integrated platform for all distribution channels. The agile teams contribute to this aim by developing a cross-channel application each. The future users of the developed application are insurance agents and sales partners. Their feedback is incorporated in the process as early as

possible, for example by conducting user interviews. This customer-centered approach is to provide higher quality and supports meeting the customers' requirements.

Lean and agile principles guide the methodology the teams use. All teams are required to use the same methodology which is Scrum extended by some Extreme Programming practices. Further, the company Pivotal provided know-how on lean start-up and agile methodologies to define a suitable methodology to be used by the teams at the large insurance company [66]. The most essential addition to the Scrum methodology in this case is the development and release of Minimum Viable Products (MVP). A MVP includes only the really necessary features, but already serves to demonstrate the value of the product. Therefore, an MVP is released very early in the process to incorporate customer feedback. This helps to focus on the most essential functions and to adapt the product according to customer requirements. It also fosters learning of the team itself [67]. After having released a first MVP after 100 days, the team extends the MVP. Further, pair-programming and test-driven development are encouraged.

3.1.2 Roles

The three teams in focus consist of experts from different functional areas. Each of these cross-functional teams has two to eight developers plus a product owner, a Scrum master as well as business analysts and an UX designer. Further, one of the more experienced developers takes on the role of the IT architect. As in Scrum, the product owner has end-to-end responsibility for the product and takes care of the product backlog of his team. The business analysts support the product owner. The Scrum master is responsible for teaching and adhering to the defined methodology. The Scrum master is a full time team member and can be one of the developers or business analysts. The UX designer is concerned with designing user interfaces and optimizing usability. As the teams have worked quite independently from each other so far, none of the defined roles is explicitly responsible for inter-team coordination. However, the three teams taking part in the case study are co-located in one room to allow simple coordination and knowledge exchange if necessary.

3.1.3 Process

As in Scrum, all work is done in sprints. Each of the teams can decide how long their sprints are. Two of the teams in focus use sprints that last two weeks, the other team defined a sprint length of one week. Sprint planning, review and retrospective take place as in Scrum. The length and frequency of these events depend on the sprint length. The backlog refinement is a weekly event. The daily Scrum takes place separately for each team. Goal and content of the named meetings correspond to Scrum methodology.

Additionally, every morning an office stand-up with all agile teams from different projects, that work at this location, takes place. It is time-boxed to a few minutes and serves to exchange information about new team members, issues that affect all teams as well as events that might be interesting for others to visit. The teams can ask for help with challenges

they face. It fosters transparency and communication between the teams. The meeting does not allow the exchange of technical and business issues in detail. Further, some CoPs were established in which members of different teams take part to synchronize the teams concerning practical issues that affect all of them. However, the CoPs which foster knowledge exchange so far did not lead to visible results or overarching functions that would be very helpful for the teams.

3.1.4 Artifacts and Tools

As in Scrum, an essential artifact is the product backlog. Each of the teams has an own product backlog which is managed by the product owner of the corresponding team. The backlog items are written down as user stories, then refined and prioritized. To manage the backlog an issue and project tracking tool is necessary, which is described in Section 3.2.6. In this case, Jira Software is used by all teams.

Further, all teams use sticky notes and similar artifacts to do some modeling or even write down their user stories and tasks. So far none of the teams practices domain modeling.

There is an overarching wiki space for the entire project which includes documentation relevant for all agile teams. Additionally, each of the agile teams has an own subordinated wiki space to document organizational, as well as technical and business issues and requirements. The wiki is utilized extensively by all teams. A description of the collaboration software for the wiki and its use cases is given in Section 3.2.6. In this case, the collaboration and wiki tool is Confluence. The major artifact of each team is its developed product. In the beginning, the different products and teams seemed to be quite independent from each other. However, some interdependency as well as similar requirements were detected.

3.1.5 Architecture

Each of the teams develops its own product and therefore is responsible for its architecture. However, some technical constraints were given to the teams by IT architects. Cross-team coordination and interdependency so far did not play a major role for architectural considerations. Then the teams started to realize that dependencies exist. However, the teams need to reach their own sprint goal which does not leave a lot of time to consider overarching issues. Additionally, enterprise architects so far did not play a major role in the development process.

3.1.6 Major challenges

In this section, the challenges the agile teams on the project as well as the enterprise architects face in the case study are described. The major challenges are listed comprehensively in Table 3.1 and explained in the following paragraphs. The framework following in the next section addresses these challenges.

	Major Challenges
C1	Dealing with dependencies between the teams
C2	Dealing with overarching functions
C3	Cross-team coordination
C4	Inter-team communication
C5	Organizational Structure
C6	Organizational Structure as soon new teams are added
C7	Scaling Scrum methodology to more teams
C8	Lack of whole product focus
C9	Lack of architectural guidance
C10	Lacking definition of the EA role
C11	Lack of common language (intra-team) communication
C12	Cultural change towards a more lean and agile mindset in the company

Table 3.1: Major Challenges

The established teams work independently from each other most of the time. However, some dependencies exist. These were detected by the teams themselves in the development process. So far, common issues and dependencies are discussed in CoPs. However, it is not clearly defined how to detect, address and reduce dependencies between the teams (C1).

Overarching functions exist that are needed by more than one team. These functions mostly include supporting functionality, such as document management or authentication mechanisms. In the current situation every team develops its own solution for overarching functions (2). However, it might be more practicable if a function which was already implemented by another team can be reused by other teams. As copying source code is an anti-pattern which should be avoided, other teams should be able to utilize already implemented functionality via APIs or events. For the development of overarching functions, it would be beneficial if the teams have an overview of what functions were already developed. This especially could be useful for newly established teams to be able to focus on the core functionality of their MVP and in the first step use already given supporting functionality.

To be able to deal with dependencies and overarching function cross-team coordination (C3) and inter-team communication (C4) are necessary. However, both should be kept as simple as possible to avoid extensive additional effort. A process is needed to define how coordination and communication can be integrated in the development process. Where possible, need for coordination and communication between the teams should be reduced. To support this, the organizational structure plays an important role (C5). The team structure is supposed to allow the teams to work as independently as possible from each other. Currently, the teams are structured around the applications they develop. However, this

leads to the already mentioned challenges. Therefore, a new idea to be applied in the case study is to structure teams around functional areas - (sub-)domains.

Especially, in the future when further agile teams that work on the same project will be established, one has to determine the organizational structure, such that the teams can work as independently from each other as possible (C6). Currently, with only a few teams working on the product, the organizational structure, cross-team coordination and inter-team communication are not extremely crucial problems. However, the mentioned challenges were detected and it is already clear that new teams will be added to the project in the future. Then it becomes very essential to determine an organizational structure that reduces interdependency between the teams. A process for dealing with dependencies - if existing - needs to be clearly defined and simple.

To address the explained challenges, it is important to find a method to scale Scrum to more than one team (C7). Currently, each team applies one team Scrum in its development process. A method for scaling is not defined so far, but it is considered as necessary not only by the teams themselves, but also by project management and enterprise architects.

As currently one team Scrum is applied, every team has an own product owner and backlog. However, this might hinder retaining whole product focus as each team only focuses on achieving its own Scrum goal. A lack of whole product focus might reduce overall product quality (C8).

The teams determine the architecture of their application themselves. This is practical for the technical architecture. However, considering the challenges concerning overarching functions some architecture guidance from a business point of view is considered to be necessary. Additionally, some more methodological guidance is needed (C9).

So far enterprise architects were not involved in the development process. Their role is not defined. However, the enterprise architects might be able to reach an overview of all teams, especially concerning dependencies. They could also provide the lacking architecture guidance from a business perspective and might be also suitable to provide methodological guidance (C10).

When talking about requirements within a team a common language is of advantage. So far no ubiquitous language was defined by the teams. Not all teams need the same language, but it is advantageous within each of the teams (C11).

Another challenge is to bring back the teams after they developed their product and received training on the agile methodologies back to the main location of the company. The teams then focus on the operation of their product. They are supposed to impact the organizational culture with their new skills and mindset change (C12). However, the process after the development of the their product is not entirely defined, as most of the teams are still working on their product. Therefore, this challenge is not in focus of this master's thesis.

3.2 Framework for (Enterprise) Architecture in Agile Teams

To begin with applying DDD, some strategic as well as tactical DDD practices are adopted in the international insurance company. Therefore, a framework for enterprise architecture is presented in this section. First, the connection between agile scaling approaches, DDD and architecture is explained. Taking into account the findings of the previous sections and some further literature concerning agile architecture and DDD, it is explored how the different concepts can be integrated and applied. Then the idea on how to make use of those practices in the case is explained. Based on these findings and ideas a framework including roles, a process, artifacts as well as tools defined. The framework itself is represented in one comprehensive graphic.

3.2.1 Agile Scaling, DDD and Architecture

"If you think that design is a dirty word when agile is in practice, it is not with DDD. Using DDD with agile is completely natural. Always keep design in check with agile. Design need not be heavy." [83]

To scale Scrum, the agile scaling approaches, Nexus, LeSS and SAFe, are considered to be helpful. However, the frameworks provide only little architectural guidance. Nexus does not provide an idea on how to evolve architecture at all. Often Scrum teams act like architecture is not relevant for agile teams, but Scrum or Nexus cannot replace architecture [70]. Simply saying that architecture emerges somehow might work for one or a few loosely coupled Scrum teams, but even with three agile teams in focus in the case study it was detected that some overall architectural guidance and inter-team coordination is necessary to not slow down the teams. While speed, flexibility and high quality are three of the main objectives of agile methodologies, with more than one team minimal architectural guidance as in Scrum or Nexus are not enough.

Therefore, LeSS and SAFe are more closely considered to be a more suitable approach for the teams in the case study in the future. Both of these agile scaling frameworks recommend using DDD practices when they address the topic of architecture. When considering DDD as a support for agile teams as well as architects, it is advantageous to distinguish between strategic and tactical DDD. While Strategic DDD especially addresses organizational structure and dependencies between the teams, tactical DDD is about supporting the teams with domain models and a common language. DDD provides enterprise architects and development teams with very useful strategic as well as tactical tools to improve their processes. "DDD is not a heavyweight, high-ceremony design and development process. DDD is not about drawing diagrams. It is about carefully refining the mental model of domain experts into a useful model for the business. It is not about creating a real-world model, as in trying to mimic reality" [83]. DDD can be applied alongside each agile development approach the team is comfortable with. The models produced are not only the domain model, but the software as well. Both are refined continuously and incrementally

over time [83].

The objectives of DDD and agile development are quite similar. This also supports the hypothesis that these approaches can be combined very well in practice. Vernon (2013) lists and explains major benefits of DDD [83]. In the following those benefits, which can also be seen as an organization's goals to reach by applying DDD, are connected to the aims of agile development. First, DDD helps to focus on what is most essential to an organization's business and success - the core domain. By modeling and context mapping the organization is not only able to determine and set boundaries around its core domain, it is also able to enhance understanding of its own business. Not only the developers increase their knowledge about the business, but also domain experts themselves. The main objective of agile development projects is to provide business value to the customer quickly. This can only be reached if business as well as software experts have a shared understanding of the product to be developed. An ubiquitous language based on the domain model supports their communication and understanding. Close collaboration between developers and domain experts is emphasized by DDD as well as in Scrum and agile scaling frameworks, where teams are cross-functional and the product owner is the connector to the customers in the business and frequent sprint reviews are used to show results to the customer and future users. DDD and agile methodologies therefore can increase user experience when combined.

Further, it is emphasized that in agile development it is best to develop products in feature teams. This is supported by DDD, through bounded contexts which help to verticalize the application landscape and to determine the organization structure. One vertical slice should correspond to one subdomain - in the best case one bounded context - and should be worked on by one agile team. This is suitable to keep the teams working as independently from each other as possible. Reducing dependencies between contexts and teams is the major characteristic of DDD to increase development speed. Developing quickly is a major aim of agile development as well. DDD can be a method for agile teams to increase their speed.

Bounded contexts determine where a model applies and where not. This means that each agile team has an own model in its context. This allows each team to evolve its model as well as its software iteratively over time. Modeling as well as development are done in an agile and iterative way.

Further, DDD helps to better organize and understand enterprise architecture. Setting explicit boundaries around contexts and understanding them, contributes to understand where and why integration between two contexts are necessary. If bounded contexts are clear, the relationships between them become clear as well. To model dependencies comprehensively context maps can be utilized. This fosters a very deep understanding of enterprise architecture [83].

Often it is argued that architecture contradicts agile values and principles. However, if architects themselves work in a more agile way, support by enterprise architects from an overarching team can be beneficial. "Our mantra for Architectural Agility is 'informed anticipation'. The architecture should not over-anticipate emergent needs, delaying deliv-

ery of user value and risking development of overly complex and unneeded architectural constructs. At the same time, it should not under-anticipate future needs, risking feature development in the absence of architectural guidance and support. Architectural Agility requires 'just enough' anticipation. To achieve the quality of being 'just enough', architectural anticipation must be 'informed'" [18]. While it is now widely spread that agility needs some kind of architecting, it is not clear how this should be done [70].

There are several questions, architects and agile teams have to ask themselves, when determining the role of architecture in the agile development process. Additionally, the answers given to those questions may vary for every organization. Table 3.2 gives an overview of important questions and possible answers to them [70].

Questions	Possible Answers
When is the activity performed?	<ul style="list-style-type: none"> • Upfront – separate phase to determine architecture • Sprint 0 – initial Sprint to determine initial architecture • Continuous – with spikes, during planning or coding • In parallel – by a separate architecture team
What is the target of the activity?	<ul style="list-style-type: none"> • Selected aspects – architecture work for “architecturally significant” aspects • Every story – is targeted by architectural work • Every epic – is targeted by architectural work • Every delta of a sprint – all changes in a sprint are targeted by architectural work • Overall product – architectural work focuses on the overall product
How is the documentation done?	<ul style="list-style-type: none"> • None – no explicit documentation at all • Knowledge – what developers know is the only form of documentation • Whiteboard – collaboratively and frequently changing in the team room • Wiki – which requires maintenance • Architecture Document – in a separate document with the major concepts and design • Models – created by the team as a documentation
Who is responsible for an activity?	<ul style="list-style-type: none"> • Complete team – everyone in the team is responsible • Separate group in team– some team members which interested & experienced in architecture • Separate role in team– a dedicated role in the team • Separate team- serves as consultant

Table 3.2: Major questions and possible answers for determining the role of architecture in agile organizations [70]

These questions and answers need to be kept in mind also when defining the role of architecture in the framework. Therefore, those questions need to be answered in the case study with the international insurance company. The critical questions address when architectural activity needs to be performed, which artifacts to target, how to document the architecture and who actually is responsible for the architectural activity.

After detecting the listed challenges and analyzing scaling frameworks, it became clear that scaling the agile teams following one of the frameworks without further additions and adaptations, especially concerning architecture, is not a suitable approach. Therefore, also DDD is seen as a possibility to contribute to solving the challenges. A first conceptual idea on how to use agile scaling in combination with DDD to incorporate architectural

guidance in the context of the large insurance enterprise is explained in the following. As DDD was not used in the insurance enterprise before, enterprise architects try to take some first steps towards establishing DDD. It was decided to get started with strategic level DDD, before some tactical practices were adopted as well. The idea was to include DDD in the development process without slowing down the teams. Therefore, the additional efforts for the teams themselves need to be kept small. What steps are taken in the international insurance enterprise and how DDD is involved in the process as well as which role enterprise architects play, is explained in the next section.

3.2.2 Overview

The framework to address the challenges of the agile teams at the international insurance company comprises the relevant roles, a process including events, artifacts and tools. The starting point for defining the framework are the current practices of the agile teams in the company. However, these practices need to be improved. Therefore, this framework is a proposition how the agile development process in the organization should be adapted to improve concerning the presented challenges. The framework addresses especially how enterprise architects can take part in the process and are able to provide value to each of the agile teams as well as the overall project.

Figure 3.1 shows the framework which is divided into three main parts: strategic DDD, the development process and tactical DDD. The first part describes how strategic DDD can be incorporated in the agile development process. The area in the center of the framework describes the development process including what has to be done within the teams and cross-team coordination. Here the main focus is mostly on scaling agile teams. The area on the bottom addresses how tactical DDD supports the development process.

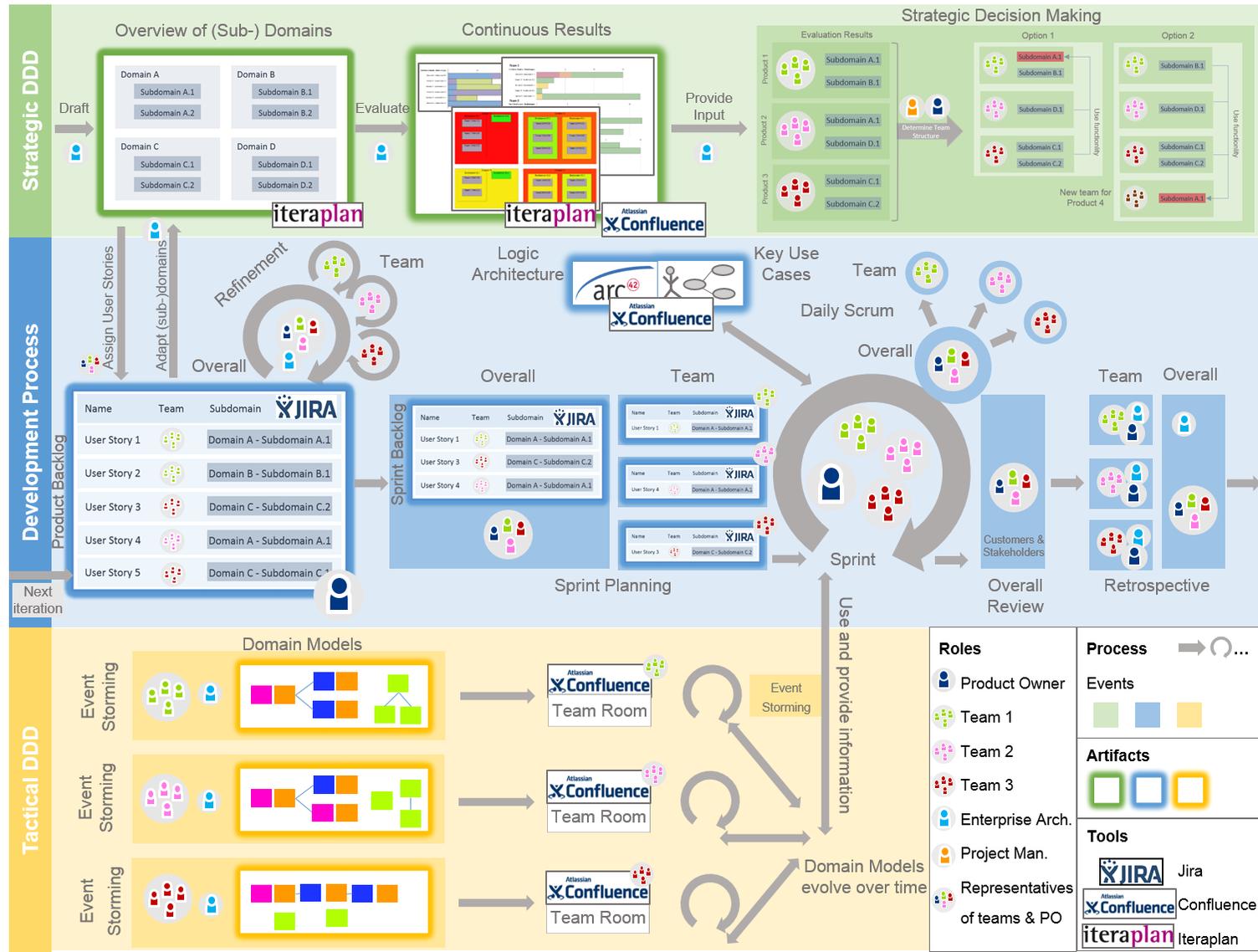


Figure 3.1: Framework for (Enterprise) Architecture in Agile Teams

In the following paragraphs, an overview of these three parts is given, before in the next section the roles, process, artifacts and tools are explained in detail. The description includes reasoning why certain responsibilities, process steps, events, tools and artifacts are beneficial. Reasons for certain recommendations were found in the already presented literature or are based on experiences made during the case study.

The framework is a suggestion how the teams should work in the future incorporating DDD in their daily work. Enterprise architects support the teams during the entire implementation of the framework and are an essential element in the framework. The strategic and tactical DDD part of the framework were already tested, partly implemented and evaluated during the last months. In contrast to that, the development process was not changed so far, but might be adapted in the future based on the framework. The already implemented elements as well as the elements that might be implemented in the future are evaluated in chapter 4 with a pre-study and interviews.

Strategic DDD The strategic DDD process in the framework serves mainly to determine in which subdomains the different teams on the project work. This is reached by assigning all user stories of all teams to the subdomain they belong to. An overview of all domains and their subordinated subdomains was created by enterprise architects before applying the defined framework. However, the overview of the domains can be adapted in the course of the process. The assignment is conducted by the teams themselves and is continuously evaluated by enterprise architects. The results support decisions of project managers and the product owner. These decisions mostly are about the organizational structure of the current teams and especially about the structure how and with which structural changes new teams are added to work on the same project.

Further, the results of the evaluation give an idea of what overarching functions might be, that are needed by all or at least more than one team, and help to detect dependencies between the teams. Based on the results of the strategic DDD process teams are structured around the subdomains they work in. Currently, they are not structured in this way, but the established process helps to define the subdomains the teams work in. Especially for each of the subdomains containing overarching functions, a new team could be established. In the large insurance company this approach will be tested on the example of one team for one subdomain. If this approach shows to be beneficial, further teams for further subdomains might follow.

This approach is mainly driven by organizational and architectural considerations. As in the agile scaling framework, especially mentioned in SAFe and LeSS, the teams are supposed to be structured based on features they build. Currently, the teams are structured depending on the components of the entire application they develop. Therefore, the strategic DDD process is supposed to make way for a more feature-centric team structure. To get started, teams should be structured depending on functional domains which leads to the development of end-to-end features in a certain functional area. A team in a certain subdomain is supposed to provide the other teams with their features. Summing up, this

part of the framework describes how an organization can get started with DDD from a strategic perspective.

Development Process The central part of the framework is the development process which is the main process of all teams and which is enriched by DDD practices. During the development process all teams give input to the DDD processes. Based on their inputs, the teams can also profit from the results of the incorporated DDD approaches. As major challenges in the case study arise from the fact that there are many teams working on one project, another focus in the development process is the cross-team communication and coordination. The development process in the framework incorporates many elements as defined by LeSS. The LeSS framework is considered to fit best to address the current challenges as the number of teams is still manageable, but likely to increase in the near future. Further, LeSS incorporates agile modeling approaches which can be easily connected to DDD. Additionally, LeSS suggests to have a single product owner and a single product backlog for all teams which supports the whole product focus. This is crucial for product quality and might help dealing with overarching functions. The process, roles, artifacts and tools are described in detail in the following sections.

Tactical DDD The tactical DDD part of the framework describes how agile teams can use the DDD approach to contribute to their development process. The central element of tactical DDD is the domain model which serves to support a common understanding and an ubiquitous language in each team. All input for the domain model comes from the respective team, while the enterprise architects mainly provide methodological guidance. The domain models are continuously improved throughout the entire development process. Tactical DDD describes an approach to domain modeling which is also cited as very helpful by LeSS and SAFe [20, 41]. For evolving the domain model, agile modeling techniques, such as event storming which is also suggested by Vernon, are used [84]. The domain model is mainly used by the team by which it is defined and evolved. However, it might make sense that two teams which work on similar tasks have an open exchange about their models. Further, before making important strategic decisions, the product owner and project managers might take these models into account as well. This can also serve to validate the results of the strategic DDD process. Summing up, this part of the framework describes how agile teams in an organization can get started with DDD from a tactical perspective in a bottom-up approach.

3.2.3 Roles

In the following the roles and their responsibilities in the framework are explained. The framework includes, next to the development teams consisting of developers and a Scrum master, a single product owner, project managers and enterprise architects. Figure 3.2 gives a comprehensive overview of the roles and their involvement in the different events

and process steps in the framework. "x" means that the role always should take part in the event, while "(x)" means that participation is optional, but would make sense depending on the current situation. The roles are described in detail in the following paragraphs.

Roles and their involvement in the process/ events	Product Owner 	Development Team 	Representatives of the Teams 	Project Manager(s) 	Enterprise Architect 
Process					
Overall Sprint Planning	x		x		
Overall Backlog Refinement	x		x		(x)
Overall Daily Scrum	x		x		
Overall Retrospective	x		x		(x)
Overall Review	x		x		
Team Sprint Planning	(x)	x			
Team Backlog Refinement	(x)	x			(x)
Team Daily Scrum	(x)	x			
Team Retrospective	(x)	x			(x)
Continuous US Assignment	x	x	x		(x)
Continuous US Evaluation					x
Strat. Decisions	x			x	
Event Storming	(x)	x	x		x
Continuous Domain modeling		x	x		(x)

Figure 3.2: Roles and their involvement in the process/events

Product Owner In the framework it is suggested that there is one product owner for all three agile teams who manages the single product backlog. This especially includes the prioritization of user stories and to assign them in cooperation with representatives from all teams to the most suitable team. The product owner's responsibility, corresponding to LeSS, is to ensure whole product focus. The product owner can be, corresponding to the current situation, supported by business analysts.

The product owner is a connector between the teams and to the customer. This means he takes part in all overall meetings, such as the overall backlog refinement, sprint planning, daily Scrum, sprint review and sprint retrospective. In these meetings next to the product owner, suitable representatives from all teams participate. This allows the product owner in the framework to communicate with all teams continuously and in parallel. This especially allows discussions about dependencies and responsibilities of the teams. The product owner decides after these discussions, which can happen in the overall refinement and sprint planning, which story is implemented by which team. The product owner, as a connector to the customer, moderates the sprint review in which representatives from all teams present what they have achieved in the previous sprint.

The product owner can take part in the team meetings - sprint planning, backlog refinement, daily Scrum and sprint retrospective - of one of the teams, if required. Most likely, the named intra-team meetings for each of the teams take place at the same time. There-

fore, the product owner can only participate in one of them. However, this is optional. It also makes sense if the product owner takes part in an event storming workshop. Further, the product owner is a connector to higher level management being in continuous exchange with the project managers. Based on the inputs from the strategic DDD part, he/she can advise the project managers concerning organizational structure. He/She should be the first to realize if there are too many dependencies between the teams. Then he/she might suggest a reorganization of the teams. Further, he/she should discover early if an additional team should be established. The product owner in general is responsible for all issues that concern all teams and the entire product.

Scrum masters The Scrum master is not depicted explicitly in the framework (and also not in Figure 3.2), as the role corresponds to the Scrum master in LeSS and has no specified inter-team responsibilities. Each team has an own Scrum master who has a deep understanding of agile development practices and coaches the teams. The Scrum master can be a developer or business analyst at the same time. He/She has no further responsibilities concerning inter-team coordination and communication than any other team member. If it is regarded as necessary and practicable after a first implementation of the framework, the Scrum masters also could be educated in DDD practices to support the other team members. This might be especially necessary if many new teams are added and enterprise architects do not have the capacity to conduct workshops with all teams.

Development Team The developers in this case study are organized in three teams. As in the different agile scaling methods, the teams are self-managing, co-located and long-lived. The teams clarify, implement and test user stories. Further, development teams should be cross-functional and cross-component feature teams. This is in the framework central as the teams should be organized around the subdomains they work in. As in the current situation, one of the more experienced developers can take on the role of an IT-architect. Further, an UX designer can be part of the team. However, this role is not depicted in the framework and optional.

In the overall meetings - sprint planning, backlog refinement, daily Scrum, sprint retrospective and review - not all team members participate, but only representatives of the teams. The suitable representatives of each team are the persons with the best knowledge in the discussed issues. If a team is not involved in a certain issue and in the sprint no synchronization is necessary, teams might not send a representative to the overall meetings. However, it makes sense in terms of coordination and communication. In the team-internal meetings - sprint planning, backlog refinement, daily Scrum and sprint retrospective - all team members take part.

For the continuous user story assignment each team can decide itself if the entire team does the assignment or only one or two team members (as in the pre-study). However, it makes sense to have a person who is responsible that it is continuously done and can

be addressed by enterprise architects for questions that might occur. For event storming workshops and domain modeling it should be determined who participates depending on the discussed issues. However, it makes sense for the teams to discuss and adapt their domain models collaboratively in their respective backlog refinement or sprint retrospective to keep it up-to-date.

Project Manager(s) The project managers, of which there are two in the project in this case study, mostly use the input from strategic DDD for strategic decisions, such as determining the organizational structure and deciding if additional teams are necessary. Therefore, they take part only in the strategic DDD process of the framework and mostly use the results of the process without giving essential input to the evaluations. However, in the context of the case study the project managers participated in a higher level workshop which was utilized to detect overarching functions which are to be implemented within the project. The results of this workshop were compared to the results of the pre-study to verify both workshop and pre-study results. Such comparisons might be especially helpful in the first time of the implementation of the framework to check the results of the assignment and their usefulness for decisions.

Enterprise Architects The enterprise architect is part of a central architectural team. Depending on the number of teams and the project complexity one or more enterprise architects can be involved. In the framework the role of the enterprise architect is depicted as one person.

The framework itself was developed by enterprise architects. Therefore, this role is critical as it provides methodological guidance to the teams. The enterprise architect in the framework is especially involved in the strategic and tactical DDD process.

On the strategic level the architect provides the overview of domains and subdomains to the teams. This includes a first draft of the overview as well as coaching the teams on how to use this artifact. Further, the enterprise architect evolves the domain overview considering the input from the teams. He/She is especially responsible that the overview is adapted accordingly in all tools. Sometimes and especially in the beginning the enterprise architect supports the teams with the continuous user story assignment. Further, this role's responsibilities comprise the continuous evaluation of the user story assignment of subdomains and presenting the results comprehensively to the decision makers which are the product owner and project managers. The enterprise architect himself has no decision making authority, but provides input to the decisions.

The same assumption applies on the tactical level. Here enterprise architects introduce the method of event storming as well as domain modeling to the teams. These approaches mostly provide value to the teams and only are useful if the teams themselves give their input. The teams need to evolve the domain models over time. The event storming approach was considered as beneficial by enterprise architects to define a first draft of the domain model, but also to further evolve it. In this case study, the enterprise architect is

the moderator in all event storming workshops. The enterprise architects also teach the teams how the event storming approach works and how they can incorporate the domain model in their development process in a way that provides value to the teams in terms of reaching a common understanding and defining an own ubiquitous language for each team. To support this the enterprise architect optionally participates in team backlog refinements or team retrospectives to help with keeping the domain model up-to-date. For the same reason he/she might also support the teams with their continuous domain modeling.

As agile scaling of the development process became a much discussed issue in the entire project, enterprise architects provide input by analyzing agile scaling frameworks and by making a proposition on how to scale the teams in the future in the form of this framework. However, this role is not a direct part of the development process in this framework, but works as a facilitator to support agile scaling.

Further, the enterprise architect has an overview of the entire project and evaluates the inputs from the DDD processes. The idea of incorporating above team level architects in the framework is based on the approach to architecture in SAFe.

3.2.4 Process

In the following the processes on all three levels, with focus on the single events, are described. This includes not only the known Scrum events as well as sprints, but also the further established events storming workshops. Further, the continuous user story assignment and evaluation are part of the process as well as strategic decision making.

Sprints All discussed agile scaling frameworks - Nexus, LeSS and SAFe - suggest to synchronize the sprints for all teams. This means the same sprint length as well as sprint start and end. Here a sprint length of one to two weeks is suggested. Only with synchronized sprints, the defined overall meetings make sense and can provide value to the teams.

All development work is done during the sprints. A sprint starts with the sprint planning, while its ends with the sprint review and retrospective. Right after the retrospective, the next sprint is started with the sprint planning. The backlog refinement takes place once in each sprint which can be seen as a preparation for the sprint planning. Further, events are the daily Scrums.

With every new sprint the development process is repeated. The strategic and tactical DDD process are running in parallel. However, on the strategic DDD level only in the beginning a draft of the overview of domains and subdomains is provided by enterprise architects, while after that this artifact is only adapted based on the input of the teams. On the tactical level not every sprint and event storming workshop is necessary. Event storming especially makes sense in the very beginning of the development process or for scoping the next MVP. If during the development process issues come up, which need to be discussed by the entire team, an event storming workshop can be helpful. Therefore, these workshops are scheduled and conducted as needed.

Backlog Refinement As events during the development process, there are the overall backlog refinement as well as the team backlog refinement. Having both an overall as well as a team backlog refinement is like in LeSS. However, in LeSS only the overall backlog refinement is optional. In the framework it should be determined based on the current user stories in the backlog if both meetings make sense or only the overall backlog refinement. Each team can decide itself when a team backlog refinement is necessary and when not.

The overall backlog refinement is done by the product owner in cooperation with representatives from the development teams. They refine user stories and prioritize them. Additionally, in this meeting the user stories which are not assigned to a subdomain should be assigned. Based on this, teams already can get a first idea, which user stories might be assigned to the for implementation.

The user stories that are likely to be assigned to a team, can be refined in the team backlog refinement. A team backlog refinement might not really be necessary, if the representatives of the teams have refined the user stories already properly in the overall backlog refinement. Therefore, this meeting is optional. However, this meeting can also be used to discuss how future user stories that are implemented might affect the domain model. Usually, all members of the development team participate in this meeting.

Sometimes it makes sense that the enterprise architect takes part in the backlog refinements. In the overall backlog refinement, the enterprise architect supports with the assignment to the subdomains and in the team backlog refinement he/she supports with adapting the domain model. In both he/she is supposed to mainly give methodological guidance. If he/she takes part or not, is discussed with the teams upfront. However, it is likely that the teams hold their backlog refinements at the same time. Then the enterprise architect needs to decide in which of the intra-team refinements he/she takes part.

Sprint Planning As a very essential part of the development process, there is an overall sprint planning as well as a team sprint planning which is conducted separately and in parallel by each of the three teams.

In the overall sprint planning, the product owner as well as representatives from all teams take part. The participants discuss shared work and decide which user stories will be implemented by which team in the upcoming sprint. The workload of the teams has to be divided depending on the number of developers and availability. Therefore, a rough estimation, e.g. using story points, is helpful. As soon as the teams are structured according to their subdomains, the user story assignment - which should have happened before the sprint planning - is a very good indication for deciding in which team's functional area the user story belongs. In the best case the user story assignment was already discussed in the backlog refinement. Further, the priorities of user stories in the single product backlog have to be considered.

The product backlog is managed in an issue and project tracking tool, e.g. Jira. In the sprint planning, in the tool a new sprint is opened and the user stories to be implemented are moved into the sprint backlog. The assignment of user stories to the teams should

have been done before the sprint planning in the tool, so that the teams are able to plan the backlog for the user stories that were assigned to them.

The team sprint planning is held by each team separately directly after the overall sprint planning. The entire development team participates in this meeting to plan the upcoming sprint. Stories should be divided into smaller tasks and the teams need to estimate the workload of each user story more precisely. The teams can use the meeting to discuss which user story is done by which team member. Further, they can discuss which impact the user stories have on their domain model.

Daily Scrum As daily events during the development process, there are the overall daily Scrum and the team daily Scrum. This proposition is similar to the daily Scrums in Nexus. In the current situation there is the daily stand-up with all agile teams from all different projects discussing mostly organizational issues. The framework suggests to do an overall daily Scrum instead in which the product owner and representatives from all teams in one project are involved. This meeting should be used to discuss the tasks the teams work on that also might affect other teams. So the focus should be on dependencies and integration issues. The meeting should be kept very short and should be time-boxed to around 10-15 minutes.

Currently, the teams conduct their own daily Scrum directly after the stand-up. This should be kept as it is with the only difference that the team members who have participated in the overall daily Scrum before, give a summary about what was discussed in the overall daily Scrum.

Sprint Review In the development process in the framework, it is suggested to have only an overall sprint review with the customer and representatives from each team who present what their team has implemented in the sprint. The meeting can be used to get feedback from stakeholders. The meeting is conducted as the review in Nexus and LeSS. This meeting is not connected to the strategic and tactical DDD process. However, when communicating with the customers which are most likely business experts, it is important to use a common language based on the language defined in the domain models of the teams.

Sprint Retrospective As a very essential part of the development process, there is an overall sprint retrospective as well as the team sprint retrospective which is conducted separately by each of the three teams. In contrast to the other meetings, for the retrospective first the team-internal retrospective takes place, before representatives of the teams participate in the overall retrospective.

The team retrospective is used to discuss what went good and bad in the previous sprint and to discuss means to improve. Further, the meeting can be used to adapt the domain model if changes occurred during the sprint that were not incorporated in the model. It also makes sense that an enterprise architect participates in the meeting. However, this

is optional and is not necessary every week. As the team sprint retrospective takes place for all teams at the same time, the enterprise architect can take part in the retrospective of another team every week.

After that, one or more representatives join the overall sprint retrospective to discuss those issues that affect more than one team. The meeting should be utilized to discuss what went good and bad concerning dependencies and integration. The sprint retrospective is held right after the sprint review. Directly after the retrospective, the overall sprint planning for the next sprint takes place.

Continuous User Story Assignment and Evaluation The assignment of user stories to the subdomains of the organization as a part of the strategic DDD process serves to determine a suitable organizational structure in line with the DDD approach. The goal is to have teams that are working in one bounded context within one subdomain.

In the framework and the pre-study it was determined that the assignment to a subdomain for each user story should be done before it becomes part of a sprint. Sometimes it might be easy to assign it, when it is created. For some user stories further refinement is necessary to be able to assign it. Then the team can discuss the assignment in the user story refinement.

The major artifacts involved in this process are the overview of domains and subdomains as well as all user stories in the overall product backlog. The artifact which arises from this, is the evaluation of the assignment. For the detailed description of the artifacts see Section 3.2.5. The continuous assignment is done in the issue and project tracking tool. For the continuous evaluation an enterprise architecture Management Tool is necessary, while for the pre-study (see Section 4.1) Microsoft Excel was used.

Strategic Decision Making The strategic decision making in the framework is done by the product owner (currently product owners) and project managers. They are mostly concerned about organizational structure in the framework. However, their focus is to make the entire project and the developed product successful. The framework contributes to this decisions about team structure. Therefore, the strategic decision making in the framework is only focused on organizational structure. This comprises not only potentially restructuring existing teams, but also deciding about the responsibilities as soon a new team is added.

In this case study, the core subdomains of the three teams differ. However, next to user stories in their core subdomain, all teams have implemented overarching functions. Overarching functions are mainly supporting functionalities which are needed - potentially in a different way - by every team. Such functions could concern authentication issues, like log-in or managing user roles as well as managing documents or transactions. Until before the case study each team has developed an own solution for e.g. a user log-in. For the example in Figure 3.3 assume that domain A includes overarching functions and that subdomain A.1, as one of its subdomains, is concerned with legitimization and therefore

every user story concerned with the user log-in would be assigned to this subdomain. The major idea is to structure the teams based on the subdomains they work in. According to DDD, this fosters independence between the teams which lets the teams work more quickly. When deciding about the responsibility of a team for one or more subdomains there are two options. Figure 3.3 illustrates the two options with an example.

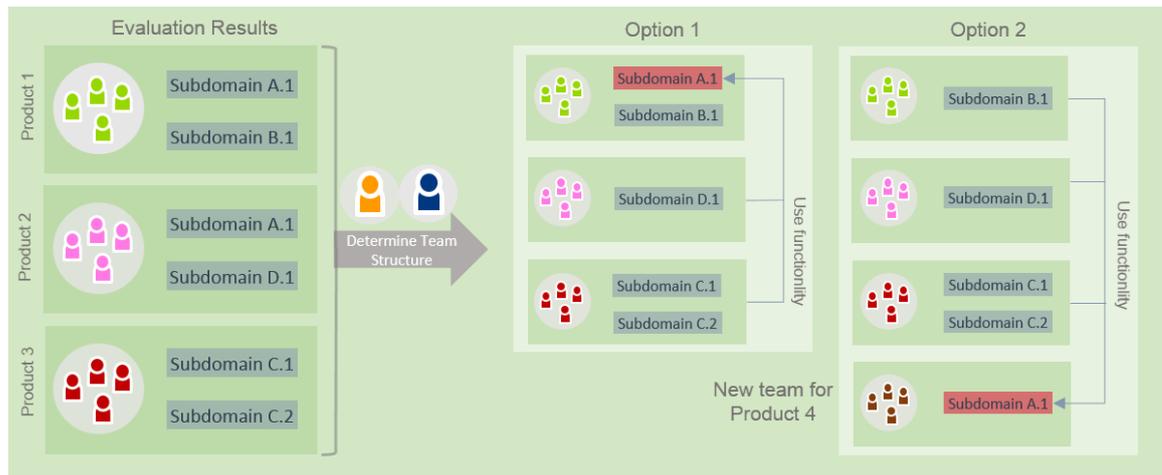


Figure 3.3: Options for the team structure with focus for providing overarching functions from subdomain A.1 to all teams and adding a new team

The first option would be to make one of the existing teams responsible for managing log-in functionality for all teams and providing this functionality via an API to all other teams (Option 1). This team would then still be responsible for its core subdomain plus one subdomain with overarching functions. In the example team 1 would be accountable for its core subdomain B.1 and the overarching subdomain A.1. team 1 has to provide functionality concerning subdomain A.1 to other teams via events or an API. Therefore, all user stories in the common backlog with subdomains A.1 and B.1 should be assigned to team 1 in the future. No new team needs to be added.

Another possibility would be to establish a new team which then is accountable for an overarching function (Option 2). This means the responsibility of the new team is to implement all user stories of subdomain A.1, while team 1 now - as the other teams in option 1 - is free to focus on its core subdomain B.1 without having to provide overarching functionality to other teams. This only works if there are a lot of user stories for subdomain A.1 so the new team is working to capacity. This option could also work the other way around, meaning that team 1 becomes responsible for subdomain A.1 and the new team works on subdomain B.1. However, this might not make sense as team 1 already gathered a lot of expertise concerning its core subdomain B.1 and therefore most likely should continue working on its core subdomain B.1.

In this case study, it was further detected that in the beginning it makes sense to let the teams start to develop their MVPs, while continuously evaluating in which subdomains they work. This allows the teams to get started quickly without being slowed down by having to provide functionality to others, while getting insights for a informed strategic decision. A strategic decision about the team structure then can be made based on numerical evidence. However, as soon as certain teams for overarching functions are established and can be provided to new teams directly when they start, the new teams are able to focus on their core subdomain from the beginning on. This will increase their development speed significantly. In the best case, there is a platform which provides the overarching functions to the teams, which is documented comprehensively so that all teams know what functionality there already is. Then the teams are able to utilize the existing functions as they are or are even able to adapt them to fit their application.

Event Storming and continuous domain modeling In DDD related architecture, it is suggested that "one of the best things you can do at the inception of a project is to use event storming" [84]. Therefore, event event storming is part of the tactical DDD process in the framework, see Figure3.4. Continuous modeling is used to refine the domain models outside of the workshops during the development process. The domain model can contribute to the development process over the entire development process. The other way around the development process contributes to the refinement of the models over time.

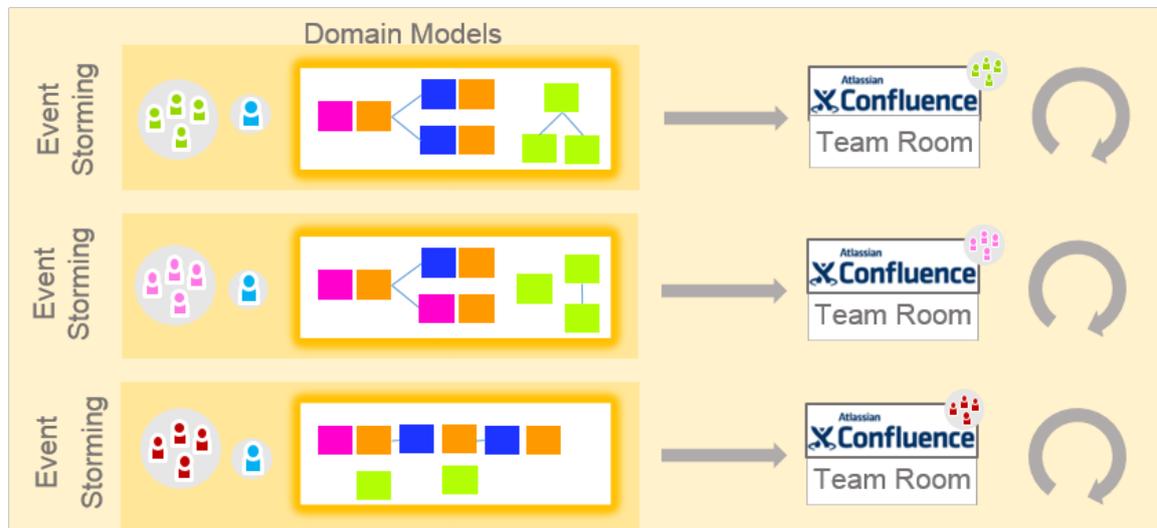


Figure 3.4: Event Storming and continuous (domain) modeling in the tactical DDD process

Event storming is a workshop format supporting the exploration of complex business domains with domain models starting with domain events as their most crucial part. It al-

lows to come up with a comprehensive model of the business flow in a domain in a couple of hours by bringing domain experts and developers together in a room to build a model collaboratively. The approach is in line with DDD, as it helps to determine bounded contexts and aggregates quickly. It has a very easy and intuitive notation that all participants can understand within a couple of minutes. The event storming workshops create an atmosphere for discussions about the business logic [15].

To understand event storming, the term of the domain events is central. A domain event is something that happens in the business domain which is of interest to domain experts. They are interested in the business domain and its logic - not databases or design patterns. Without specifying a certain implementation, events capture what happened in a domain [61]. Therefore, domain events are formulated in the past tense. In Figure 3.5 an example for business logic including similar to creating backlog item is shown [84]. An example for an event would be backlog item created.

It is crucial to invite the right people to the workshop including the right amount of people who ask the right questions and those who know the answer. The group might consist of the six to eight developers and business experts plus a moderator. Having enough modeling space is crucial. Therefore, a lot of whiteboards or other modeling space on the walls of the room are necessary. Modeling space should not be a limiting factor [15].

The first step is to collect all domain events on sticky notes. In event storming a orange sticky note is used for every domain event. As the focus is on the process within the domain the events are ordered according to their occurrence over time from left to right. If it is not clear when a domain event happens, the order can be changed whenever it the workshop the time order becomes obvious. If there are domain events happening in parallel or alternatively to each other, they can be placed under each other. The events which are not entirely clear at his point and require further discussion, should be marked with question marks or red sticky notes. These are the events the workshop have to invest the most time in to clarify them. The granularity of the model depends on the goal of the workshop. Domain events that are very important in the core domain should be modeled more fine-grained [84]. In the next step, the origin of the domain events need to be explored. Some events happen because of a user action. These are called commands and are modeled with a blue sticky note directly in front of the event they cause [15]. If it is important which person made a command, the role of the person should be written down as well - either on the blue sticky note or on a separate (yellow) sticky note [84]. An example for a command in Figure 3.5 would be, e.g. create backlog item. The person to initiate the command is the product owner in this figure. Other events are the consequences of something that happens in an external system or because of time passing. These are modeled with a purple (or pink) sticky note in front of the respective events. Both are formulated as verbs in present tense. Further, an event can cause another event. Then the two events are placed next to each other [15].

The next step is to find the aggregates (or in some cases entities) on which the commands are executed leading to a domain event. Aggregates hold the data used to execute commands and to emit domain events. After having focused on the business process before,

data has to be considered as well. The aggregate names can be written on yellow (in our case green) sticky notes. Aggregate names are nouns, such as backlog item in Figure 3.5. It is very likely that the same aggregates are used by multiple times. During the process new domain events may be found which should be added. If some aggregates are too complex, they might need to be modeled more detailed [84]. Later in the case study it was tried to model the relationships between the aggregates/ entities.

The next step is to draw arrows to represent a flow. Further, the participants should explore boundaries. Boundaries can be between areas where different models are in play. Indicators that boundaries are needed could be if different departments are concerned with the events or that different domain experts have different, conflicting definitions for certain terms. One can use solid lines for bounded contexts and also add subdomains to the picture in the form of dashed lines [84]. The concept of different artifact views was not used in our event storming workshops.

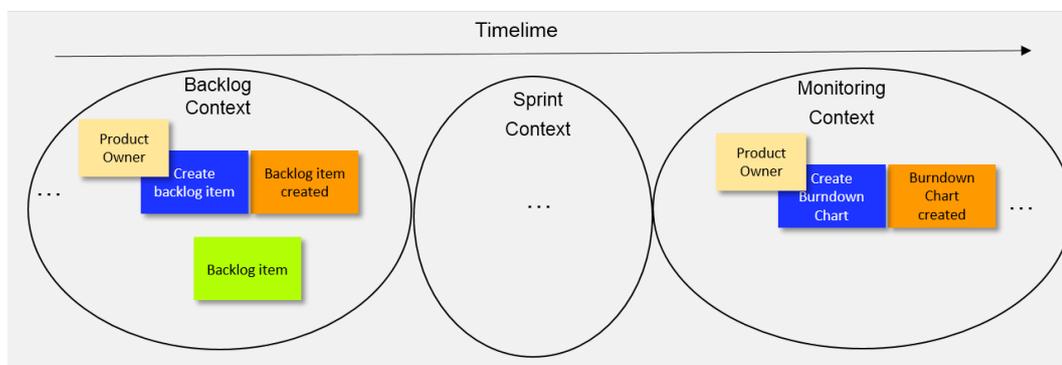


Figure 3.5: Event Storming Example [84]

This workshop approach was tested with all teams. In this case the enterprise architects were the moderators and facilitators in the workshops. After trying the approach with only one person from each team to get an initial model, further workshops are conducted with more participants to refine the model and to allow discussions. The results of the workshops were made available in the wiki for each team. This is explained in detail in Section 3.2.6. Further, it was seen as beneficial if the models with all their sticky notes are kept on the walls of the team room. However, this was not always possible because of the limited space.

In the case study, event storming is regarded as a first step towards defining a domain model with the central concept of events. There were also further tries with domain modeling starting with entities. However, the approach around entities lead to a model very similar to a data model, which did not provide much value to the teams.

The event storming workshops especially make sense in the initial sprints of a project as well as upfront or in a sprint 0 which could be used to explore the scope of the product

to be developed as well as its potential architecture starting from a business view. As the teams in the case study are not right in the beginning of their development process, the workshops are mostly done to explore benefits of the event storming approach. It was observed that it helped to detect issues that are not clear to the whole team or still need to be determined. It fosters open discussion between team members with a graphical representation on the wall which can be adapted flexible during the discussion in the workshop. After one of the workshops a member of team 1 came up with the idea to conduct an event storming workshop moderated by enterprise architects during the concept phase for his team's next MVP. In particular, the suggestion is to plan a workshop with representatives from team 1 and 2 as for the next MVP many interfaces between the two applications need to be developed. The idea is to get an understanding of the business logic of the respective other team and to check if their languages are the same and certain concepts in their applications differ. This might help both teams in the development of their next MVP and supports information exchange and communication between the teams.

The agile teams are encouraged by enterprise architects to use domain modeling continuously during the development process. It does not always need to be an event storming workshop, but after having an initial model the domain model can be useful for refining user stories or to discuss business issues which are not entirely clear within the team. Therefore, the framework suggests to look at the domain models in the team-internal backlog refinement to explore the effects of a user story on the model and the application. Further, the models might be also used in the team retrospective or sprint planning to analyze the impact of a new user story. In general, the teams can utilize domain modeling whenever it is helpful for them.

3.2.5 Artifacts

Essential to the framework are different artifacts which are used and created in the process. While on the strategic DDD level the overview of the subdomain as well as the results of the user story assignment are essential, on the tactical DDD level the domain models are the central artifacts. User stories, product and sprint backlog are very essential in the strategic DDD as well as in the development process. It makes also sense for the teams to define key use cases and document the logic architecture.

User Stories, Single Product Backlog and Sprint Backlogs Consistent with the current situation the teams define their requirements as user stories. All teams use Jira Software which is explained in detail in Section 3.2.6. Opposing to the current situation, the framework suggests that the three teams which are part of the case study should share one common product backlog including all of their user stories. As soon as it is clear which team is responsible for implementing which user story the user stories are marked with the respective team name. This supports the possibility to filter for user stories of the own team while still being able to get an overview of all user stories of the entire project. While the single product owner is responsible for these artifacts, he/she is supported by business

analysts and developers who write and refine user stories as required.

The backlog is the most central artifact for the development process and the strategic DDD process, as it includes all user stories and their assignment to a subdomain.

Overview of (Sub-)Domains The overview of the domains is the first artifact to be defined before starting to apply the framework. However, the domains and subdomains evolve over time. In this case study, a domain overview including four domains with two subdomains each is used as an example (see Figure 3.6).

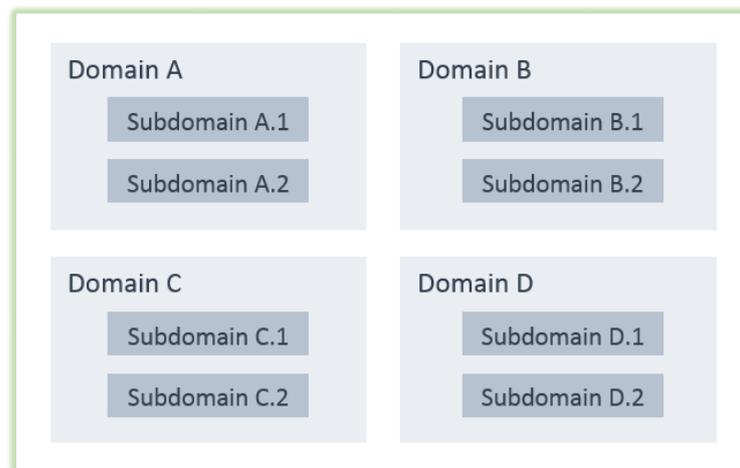


Figure 3.6: Overview of domains and subdomains

The defined artifact including the domains and subdomains of the large insurance company is very similar to one of the most central enterprise architecture management artifacts - the business capability map. "Business capabilities focus on the enterprise's ability to perform activities from a functional perspective" [36]. The business capabilities often are structured into functional areas or domains. In our example they are divided into domains with subdomains on the second level followed by the business capabilities on the third level. However, for the application of the framework in the first place the business capabilities are too granular. Therefore, the user stories are assigned to subdomains which are less granular than the capabilities.

Domain Evaluation The domain evaluation is very essential for the strategic decision making process step in the framework. The most central aspect of the user story assignment to be evaluated is the number of user stories assigned to a (sub-)domain overall and per project. This helps to detect the core subdomain a team is working on, e.g. sales. It further supports to see on which additional supporting subdomains the teams work. Here the focus is especially on overarching functions. If all teams implement user stories from

one overarching subdomains with a high number of user stories in total, this can be an indicator that one has to adapt the organizational structure and responsibility for an overarching subdomain. Further, one can take into account for these decisions the status of the user stories. Functions that have already been implemented could be provided to other teams and function not implemented yet could be implemented by a potential new team with focus on the respective subdomain. Further, one has to pay attention that there might be design and purely technical user stories which should not be taken into account for the evaluation. In the case study, there was one domain which only includes technical or IT aspects. All technical user stories can be assigned to this domain - in our example domain D (see Section 4.1) - and can be included or excluded in the evaluations as required. However, the proceeding of the evaluation in the tools is explained very detailed in the section 3.2.6. This description includes how to do the evaluation in the tools and some examples for the graphical results are presented.

Domain Model The most essential part of the tactical DDD process is domain modeling. The goal is that every team defines and evolves a domain model making the business logic in their subdomain explicit. Before defining the framework, two different approaches to domain modeling were tested.

One of the approaches was to start with the most essential entities in a domain. This lead to more data-centric model, meaning that mostly the entities were defined plus some of their attributes. This bears the risk that the teams get lost in detail defining all attributes belonging to an entity without considering the business logic which is most essential. For attributes, which are not as essential to the business logic, it is considered to be enough to be documented in code. Further, some teams have documented the attributes in their wiki already.

Therefore, another approach to domain modeling was tested - event storming which was explained in Section 3.2.4. This approach lead to much more helpful results. Collecting the essential events helped the teams to reconsider the business logic and process. Further, the commands support understanding why certain events happen. Personas and artifacts also contribute to a deep understanding of the business logic. It was also detected that with this approach in a workshop with some team members after approximately two hours the draft of a first model was ready. These models are then to be refined in additional workshops or by the teams. Considering the building blocks presented in Section 2.7.5, the event storming helps to determine domain events, some entities and aggregates. Further, it helps to find boundaries of subdomains or bounded contexts. Over time the domain models can be extended by further building blocks.

Opposing to the case study, it makes sense to start with domain modeling already in the very beginning of a project - in a sprint 0 or even before. It might help to determine the scopes and most essential business logic of the team and can be a very good basis for discussing open topics concerning the logic.

Key Use Cases and Logic Architecture Keeping a domain model is in line with agile modeling, of which the main focus is to model to have a conversation. In agile modeling, one can differentiate between models "to keep" and models formulated temporarily to be thrown away when they are not needed anymore. Figure 3.7 shows these kinds of models - keeps and temps - and how to include them in the agile development process.

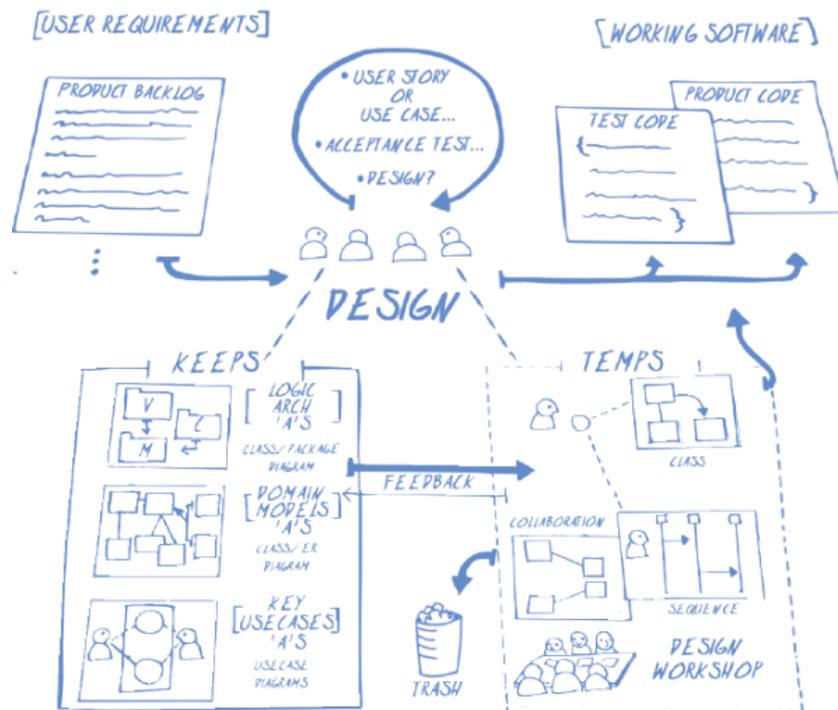


Figure 3.7: Agile Modeling for agile teams [39]

The models "to keep" describe the big picture of a system visually. Next to domain models, it is suggested to keep two other kinds of models. Logic architecture models describe the idea of the entire system roughly and key use cases can be used to understand what a typical user expects from a system, how a user interacts with the systems and how a user benefits from it [39].

The logic architecture model and key use cases are part of the framework in the development process as they are not related to DDD. The architects in the case study so far did not support the teams with these models. However, the teams started to define key use cases and started to document the logic architecture according to arc42 in their wikis. Arc42 "provides a template for documentation and communication of software and system architectures" [77]. It is based on practical experiences and independent from the used development method. However, it suits to lean and agile processes very well [77].

3.2.6 Tools

The most central tool for the framework is the issue and project tracking tool Jira Software to manage the product backlog. The teams further utilize a collaboration and knowledge management tool as a knowledge base, for collaboration and documentations. Additionally, an enterprise architecture Management tool is necessary to be able to automatize the evaluation of the user story assignment. In the following the used tools with focus on the ones, that the agile teams utilize in the case study, are described in general and how the framework suggests to engages these tools.

Jira Software As an issue and project tracking tool, the teams in this case study use Jira Software (by Atlassian) which they utilize for managing their backlogs. However, other organizations might use a different tool with similar functionality. Jira Software allows flexible planning for agile teams - with different methods, such as Kanban and Scrum - estimating the effort for user stories and ordering items according to their priorities. Further, transparent execution including the assignment of user stories to team members and a status concept is supported by the tool. There is reporting functionality to monitor progress. Backlog item types and fields remain adaptable throughout the process to support agile project management [11].

The teams already have been working with Jira before establishing the framework. Jira is the most common tool for agile software teams to manage their product and sprint backlog. For an example of a backlog in Jira see Figure 3.8.

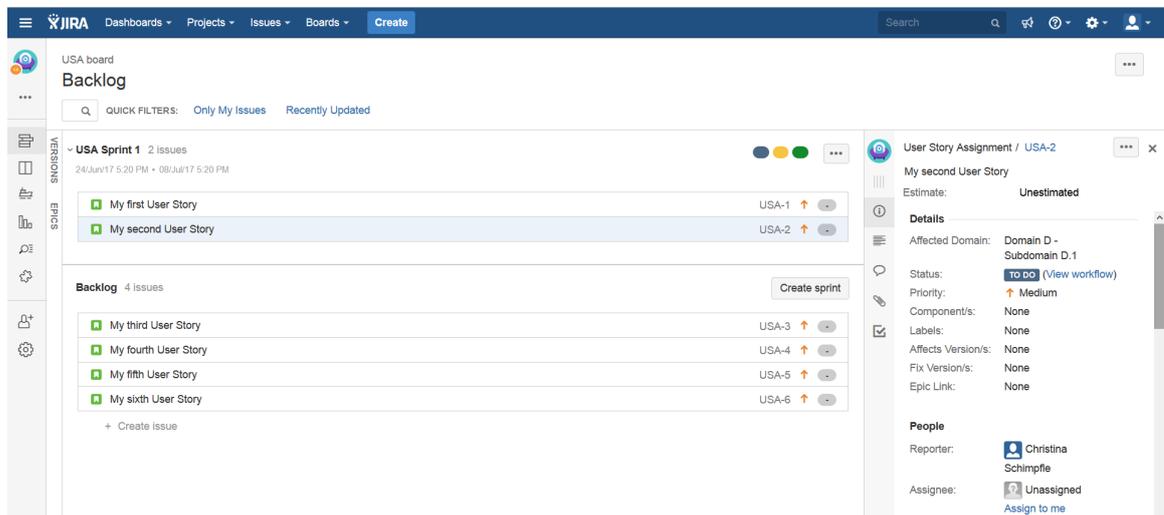


Figure 3.8: Example of a Jira Backlog including a sprint and the detail view of one user story

It is necessary that an additional field is added to the Jira template for writing down user stories. This field is used to document the domain and subdomain a user story affects. Therefore, this new field is called "Affected Domain". It consists of two drop-downs of which the first one allows to select the domain. Based on the choice for the first drop-down in the second drop-down only the subdomains of the selected domains are electable. For example, if a user chose Domain A in the first drop-down in the second one he/she can only choose subdomain A.1 or subdomain A.2. The field then includes, if subdomain A.1 was chosen, "Domain A - subdomain A.1". See also figures 3.9 for the domain and subdomain selection in Jira.

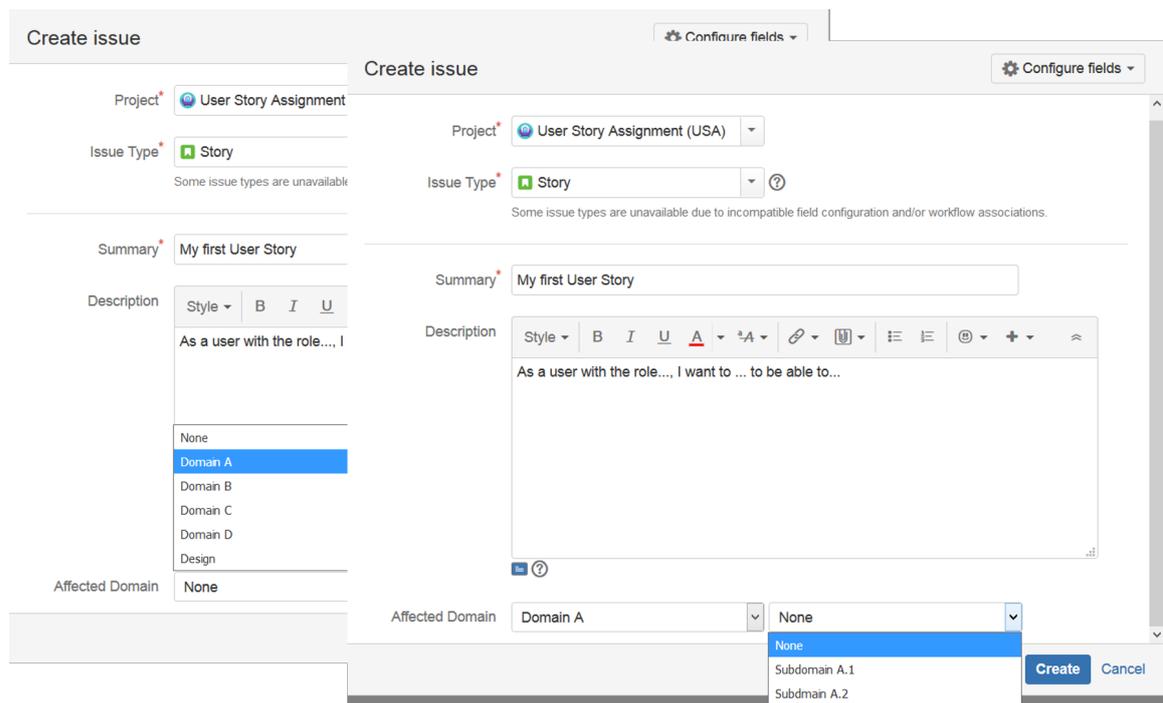


Figure 3.9: Selecting the domain and then the subdomain of a user story

Further, it is necessary to add a field to be able to assign certain user stories to certain teams, as the teams share a single backlog. To allow each team a better overview of their user stories in the next sprint or in the backlog, one can define customized filters only to show the user stories for a certain team, while still easily being able to access the view including all teams.

Jira allows exporting all user stories in a .xlsx file including value for all fields, such as for "Affected Domain". This is the base for the evaluation of the assignment in the enterprise architecture management tool, which is used for the continuous evaluation, and in Microsoft Excel, which is used for the evaluations in the pre-study. Currently, the export

has to be conducted for all teams separately as they use different backlogs. However, this will be more simple as soon as the teams utilize a shared backlog.

Iteraplan For the enterprise architects at the large insurance company Iteraplan, as their enterprise architecture management tool, is a very essential tool that is necessary in the framework to be able to automatize the evaluation of the user story assignment. Iteraplan by Iteratec GmbH is an EAM tool which can be used to plan and analyze architectures. It includes a repository to document business domains, business capabilities and business processes as well as project. Further, it can be used to document IT-services, applications, infrastructure elements and technical domains. In the tool many helpful visualization, such as (nesting) cluster graphics, portfolio, information flow and landscape plans as well as pie and bar charts, can be created [45].

Iteraplan is the tool which is used by the enterprise architects in the large insurance company. Therefore, it is an obvious choice for the architects to use this tool for the evaluation of the user story assignment. Another plus for iteraplan as a tool for the evaluation is, that all domains and subdomain were documented in the tool before the start of the user story assignment and that they are always up-to-date in the tool. Therefore, in the following a concept for the evaluation of the user story assignment is presented.

Therefore, if not already done, the first step for starting the strategic DDD process is to document all domains including their subdomains in Iteraplan. This can happen manually or via an import using an Microsoft Excel file (.xls). Assigning the same attribute to all necessary business domains is helpful, to later be able to only use these for the evaluation of the user story assignment. See Figure3.10 for all domains and subdomains in the case study as well as their respective attribute "Modellbasis = User Story Mapping MA CS". All further attributes, which are necessary for the evaluation and are described in the next paragraphs, need to be created before importing the user stories.

Further, all required teams need to be created as a project. To make clear that these teams belong to the same project, the teams are created as sub-projects of the superordinate project. For the projects, see Figure3.11.

Before one can import the files, that were exported from Jira, they have to be formatted and ordered as the Iteraplan import template requires. A template of the metamodel showing how the file needs to look like, can be exported from Iteraplan. The formatting of the Jira export files to fit the Iteraplan metamodel is for now done by hand, but will be automatized later. The user stories are imported as business processes as this element in the metamodel is related to projects and business domains. The following Figure3.12 shows a file to be imported in Iteraplan with user stories. The columns from left to right include an ID, which will be set in Iteraplan after the import, name and description of the user story, the name of the person working on the story, the attribute "Modellbasis = User Story Mapping MA CS", priority, sprint number and status of the story. All these attributes need to be created in Iteraplan before importing the actual data. The last column would include a name representing the hierarchy of the elements. However, this is not relevant here, as a

3.2 Framework for (Enterprise) Architecture in Agile Teams

Suchen

Zurücksetzen und alle anzeigen

14 Fachliche Domänen gefunden

Fachliche Domäne	Hierarchischer Name	Beschreibung	Modellbasis	Aktionen
Keine		Platzhalter für User Story Verortung	User Story Mapping MA CS	
Domain B			User Story Mapping MA CS	
Subdomain B.1	Domain B : Subdomain B.1		User Story Mapping MA CS	
Subdomain B.2	Domain B : Subdomain B.2		User Story Mapping MA CS	
Domain C			User Story Mapping MA CS	
Subdomain C.1	Domain C : Subdomain C.1		User Story Mapping MA CS	
Subdomain C.2	Domain C : Subdomain C.2		User Story Mapping MA CS	
Domain D			User Story Mapping MA CS	
Subdomain D.1	Domain D : Subdomain D.1		User Story Mapping MA CS	
Subdomain D.2	Domain D : Subdomain D.2		User Story Mapping MA CS	
Domain A			User Story Mapping MA CS	
Subdomain A.2	Domain A : Subdomain A.2		User Story Mapping MA CS	
Subdomain A.1	Domain A : Subdomain A.1		User Story Mapping MA CS	
Design		Platzhalter für User Story Verortung	User Story Mapping MA CS	

Figure 3.10: Domains and Subdomains in Iteraplan

4 Projekte gefunden

Projekt	Hierarchischer Name	Beschreibung	Aktionen
Project MA CS		Beispielprojekt für die Master Thesis von Christina Schimpfle	
Team 1 MA CS	Project MA CS : Team 1 MA CS	Beispielprojekt für die Master Thesis von Christina Schimpfle	
Team 2 MA CS	Project MA CS : Team 2 MA CS	Beispielprojekt für die Master Thesis von Christina Schimpfle	
Team 3 MA CS	Project MA CS : Team 3 MA CS	Beispielprojekt für die Master Thesis von Christina Schimpfle	

Figure 3.11: The project and the three teams in Iteraplan

user story has not got a superordinated element.

Fachliche Funktion (FF)								
Id	Name	Beschreibung	Bearbeiter	Modellbasis	Priorität	Sprint	Status	Full parent name
	USA-1	My User Story	Someone in Team 1	User Story Mapping MA CS	Medium	Sprint 1	Fertig	USA-1
	USA-2	My User Story	Someone in Team 2	User Story Mapping MA CS	Medium	Sprint 1	Fertig	USA-2
	USA-3	My User Story	Someone in Team 3	User Story Mapping MA CS	Highest	Sprint 1	Fertig	USA-3
	USA-4	My User Story	Someone in Team 1	User Story Mapping MA CS	Lowest	Sprint 1	Fertig	USA-4
	USA-5	My User Story	Someone in Team 2	User Story Mapping MA CS	Medium	Sprint 1	Fertig	USA-5
	USA-6	My User Story	Someone in Team 3	User Story Mapping MA CS	Low	Sprint 1	Fertig	USA-6
	USA-7	My User Story	Someone in Team 1	User Story Mapping MA CS	High	Sprint 1	Fertig	USA-7
	USA-8	My User Story	Someone in Team 2	User Story Mapping MA CS	Medium	Sprint 1	Fertig	USA-8
	USA-9	My User Story	Someone in Team 3	User Story Mapping MA CS	Medium	Sprint 1	Fertig	USA-9
	USA-10	My User Story	Someone in Team 1	User Story Mapping MA CS	Medium	Sprint 1	Fertig	USA-10
	USA-11	My User Story	Someone in Team 2	User Story Mapping MA CS	Medium	Sprint 2	Fertig	USA-11

Figure 3.12: Importing the user stories in Iteraplan

In the same import file, one has to determine the relations of the user stories to subdomains and projects. This is presented in Figure 3.13.

As it is not possible in Iteraplan to calculate the number of user stories assigned to a

FD-FF		FF-PROJ	
Zugewiesene Fachliche Domäne	Enthaltene Fachliche Funktion	Betroffene Fachliche Funktion	Betreffendes Projekt
Subdomain A.1	USA-1	USA-1	Team 1 MA CS
Subdomain A.1	USA-2	USA-2	Team 2 MA CS
Subdomain C.1	USA-3	USA-3	Team 3 MA CS
Subdomain A.1	USA-4	USA-4	Team 1 MA CS
Subdomain A.1	USA-5	USA-5	Team 2 MA CS
Subdomain C.2	USA-6	USA-6	Team 3 MA CS
Subdomain A.1	USA-7	USA-7	Team 1 MA CS
Subdomain A.1	USA-8	USA-8	Team 2 MA CS
Subdomain C.1	USA-9	USA-9	Team 3 MA CS
Subdomain C.2	USA-10	USA-10	Team 1 MA CS
Subdomain A.1	USA-11	USA-11	Team 2 MA CS

User Stories and assigned subdomains

User Stories and assigned teams

Figure 3.13: User stories and their assigned subdomains and projects for the import in Iteraplan

subdomain or domain, the attribute number of user stories (“Anzahl User Stories”) was created before the import. This number should be calculated when formatting the Jira export file by an Microsoft Excel tool. Then it can be imported as shown in Figure3.14.

Fachliche Domäne (FD)					
Id	Name	Anzahl	Modellbasis	Übergeordn	Full parent name
12685	Domain B	19	User Story Mapping MA CS		Domain B
12323	Keine	0	User Story Mapping MA CS		Keine
12686	Domain C	34	User Story Mapping MA CS		Domain C
12691	Subdomain B.2	0	User Story Mapping MA CS	Domain B	Domain B : Subdomain B.2
12689	Subdomain A.2	0	User Story Mapping MA CS	Domain A	Domain A : Subdomain A.2
12694	Subdomain D.1	22	User Story Mapping MA CS	Domain D	Domain D : Subdomain D.1
12692	Subdomain C.1	15	User Story Mapping MA CS	Domain C	Domain C : Subdomain C.1
12688	Subdomain A.1	38	User Story Mapping MA CS	Domain A	Domain A : Subdomain A.1
12690	Subdomain B.1	19	User Story Mapping MA CS	Domain B	Domain B : Subdomain B.1
12684	Domain A	38	User Story Mapping MA CS		Domain A
12695	Subdomain D.2	10	User Story Mapping MA CS	Domain D	Domain D : Subdomain D.2
12693	Subdomain C.2	19	User Story Mapping MA CS	Domain C	Domain C : Subdomain C.2
12687	Domain D	32	User Story Mapping MA CS		Domain D

Figure 3.14: Importing the (sub-) domains in Iteraplan including their number of assigned user stories

After all data is imported into Iteraplan, the user story assignment can be visualized. Mostly bar charts and nesting cluster graphics are useful in this case. Figure 3.15 shows the number of user stories of all projects assigned to the subdomains in a bar chart (design user stories not included).

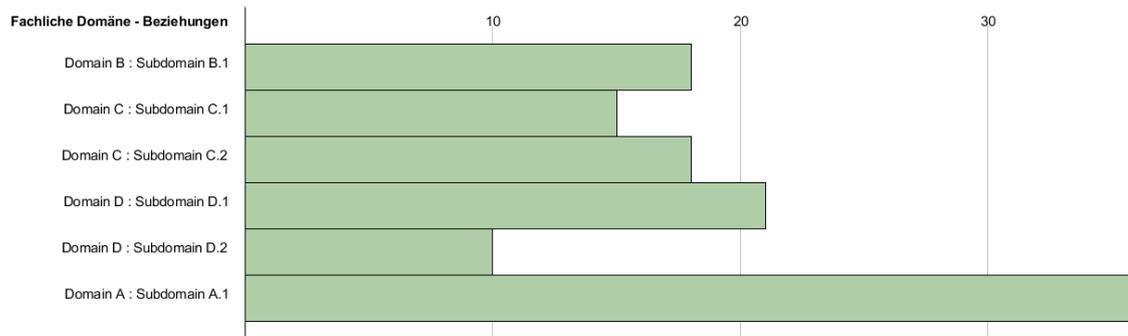


Figure 3.15: Bar Chart: Number of User Stories from all three projects assigned to the subdomains

A bar chart can also be colored depending on certain attributes, such as status of the user stories. Further, Figure 3.16 shows one bar chart per project with bars colored depending on the status of the user stories in the subdomains.

Later it was detected that simply assigning the user stories to projects or teams limits the possibilities for evaluation. Therefore, to allow more visualizations an additional attribute "Team MA CS" was created with the option "Team 1 MA CS", "Team 2 MA CS" and "Team 3 MA CS". This means that the information to which team a user story is kept redundantly. Assigning the attributes correctly was easily achieved by a bulk update in Iteraplan which was used to assign e.g. the attribute value "Team 1 MA CS" to all user stories that were already connected to project "Team 1 MA CS". To generate more comprehensive evaluations with the different teams represented by different colors as in Figure 3.17, this is necessary. Finally, the tool, as presented in Figure 3.18 also allows to present the results as a nesting cluster graphic including all domains and subdomains on the outside. The graphic further shows which teams work in which subdomains. The coloring was used to show how many user stories in total exist in the respective domains and subdomain from green (none) to red (up to 38). For example, domain A includes the most user stories of which all are concentrated in subdomain A.1 worked on by all teams, while subdomain A.2 includes no user story at all.

Summing up, these figures can be provided to project managers for strategic decision making as presentations. Later the results should automatically be displayed in the wiki for all teams as well as the project managers to see continuous results. However, this is not implemented yet.

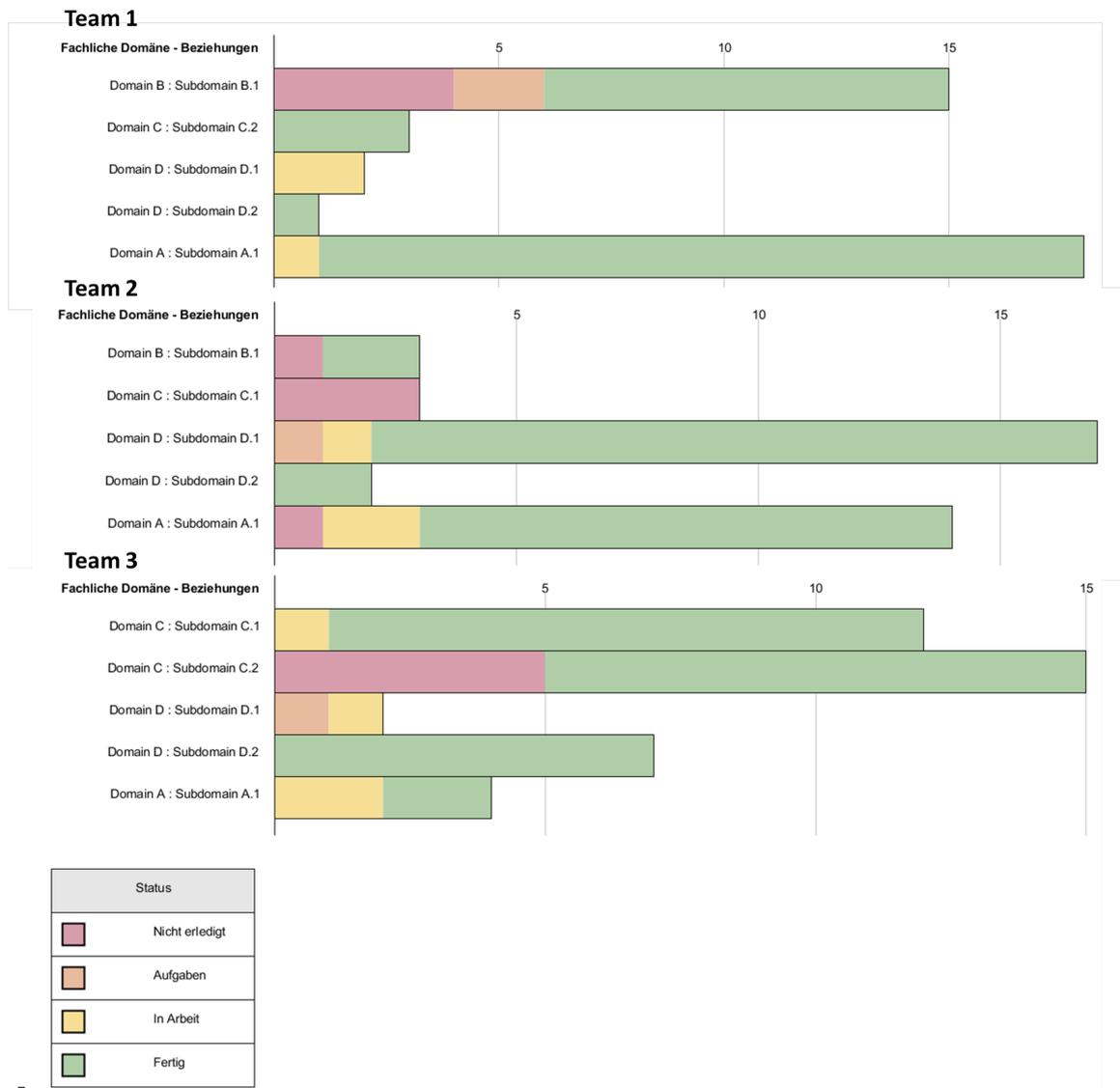


Figure 3.16: Bar Chart: Comparison of the projects depending on the number of User Stories per subdomain

Confluence Wiki Further application of the framework requires a team collaboration software and a wiki. In the case study, the agile teams utilize Atlassian Confluence as a knowledge base, for collaboration and documentations. All teams have established wiki which they use frequently. Atlassian Confluence is a tool for teams to create content collaboratively. It can be used for project collaboration e.g. by creating roadmaps, writing down

3.2 Framework for (Enterprise) Architecture in Agile Teams

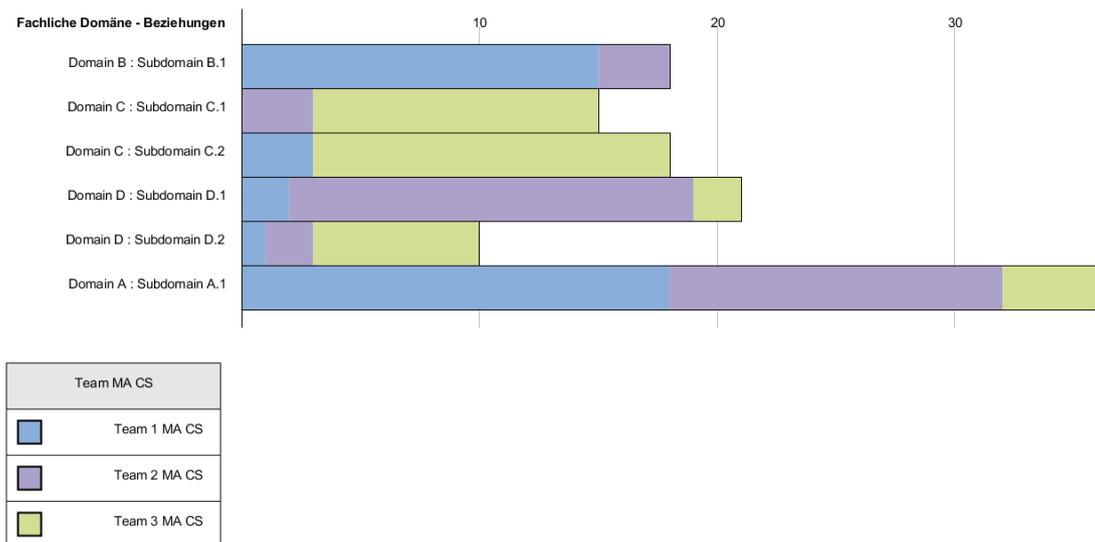


Figure 3.17: Bar Chart: Number of User Stories from all three projects (in different colors) assigned to the subdomains



Figure 3.18: Nesting Cluster Graphic: Subdomains and the teams working on them with (sub-)domains colored depending on their total number of user stories

goals as well organizational topics to share them with the entire team while also allowing all team members to contribute. Further, it can be utilized as a knowledge base or wiki.

Additionally, it can be helpful for documentation. It guarantees to have a single source of truth and transparency [10].

The teams do all documentation in their wiki. While each team has an own wiki space, next to that an overarching wiki space exists. The overarching space includes, next to other topics, documentation on strategic and tactical DDD as well as documentation of the overview of domains and subdomains. Later here, the continuous results of the user story assignment is to be presented as well.

The team-internal wiki spaces are structured roughly in the same way for all teams. The parts described in this thesis and included in the wiki are the domain model, as well as key use cases and logic architecture corresponding to the arc42 template. Most importantly on one page the event storming method is explained and a picture of the current domain for the corresponding team is included. Further, all former versions of the domain model are included to document its development over time.

4 Evaluation

The evaluation of the framework consists of two parts. First, a pre-study was conducted to evaluate if the user story assignment to the subdomains as a part of the strategic DDD process is possible and a reliable source to make assumptions about the functional domains the teams work in. Further, some interviews were conducted to evaluate the defined framework. The questions and results of the interviews concerning the framework for (enterprise) architecture in agile teams are presented. Finally, the key findings are summed up comprehensively.

4.1 Pre-Study

The strategic DDD part of the framework is constituted of the user story assignment, its evaluation and decisions based on the results. After defining the framework, the assignment of user stories to subdomains, as a central component of the framework, needed to be evaluated in terms of practicality as well as reliability and validity of its results. Therefore, a pre-study was conducted with the three agile teams. The pre-study was done in a period of two months. During that time all user stories of the three teams were assigned to subdomains. This includes not only the stories in the current sprint and in their product backlogs, but also those stories that the teams had already implemented before the pre-study started.

The pre-study has several objectives. First, it serves to evaluate if the assignment of user stories to the subdomains can be done unambiguously. Further, it helps to detect what kinds of user stories cannot be assigned to any domains. Additionally, it is important that the procedure itself is simple and not time consuming for the agile teams. If it would take the teams a lot of time, it is likely that the approach will not be accepted by them. This is especially critical in this case because the results of the user story assignment are mainly used by project management to recognize a need for action and to make decisions, e.g. concerning team structure. However, the results of the pre-study are also shown to representatives of the teams to keep them updated and to discuss the results and the next steps with them. Finally and most importantly, it needs to be checked if the approach leads to plausible results that can be used as a basis for decisions.

The basic data of the pre-study is presented in the following Table 4.1.

Basic Data	
Involved teams	Team 1, Team 2, Team 3
Timeframe	2 months
Total Number of user stories assigned	425
Roles	EAs for methodological guidance 1-2 representative(s) from each team
Artifacts	Overview of (Sub-)Domains User Stories Evaluation Template/Results
Tools	Jira Software and MS Excel (later/ in framework Iteraplan)

Table 4.1: Pre-Study Basic Data and Results

The procedure of the pre-study is described in the following. Enterprise architects had the idea for applying a domain-driven approach in the framework and for conducting the pre-study. Therefore, enterprise architects mostly work on the methodological approach to the pre-study. Beforehand, it is necessary for enterprise architects to define and overview of the domains and their subordinated subdomains of the organization. Here a first draft might be enough, as during the user story assignment certain drawbacks of the domain overview might be detected, e.g. missing subdomains or not unambiguously defined subdomains. With the input from the user story assignment the domain overview can be adapted and refined. To get started, enterprise architects conduct a workshop with representatives from the agile teams, most likely the business analysts, product owners (at the time of the pre-study each team has its own product owner) and/or Scrum master to introduce them to the approach and the overview of the domains and subdomains. After that it is necessary to get to know how the agile teams themselves work. In this case study and in the framework, it is central that the teams formulate requirements as user stories and organize them in Jira Software. Therefore, enterprise architects use an approach considering this knowledge. This means that the teams don't need to make changes to their development process or use additional tools. This might be very important for the acceptance of the approach. To make use of these characteristics the approach is to utilize the current version of the overview of the (sub-)domains as well as the user stories the teams implement to determine in which domains or subdomains the teams work. The tooling for the pre-study as well as later in the framework is Jira which the teams use for the assignment. Further, they are provided information about the overview of domains and subdomains in the Wiki.

After having decided which teams take part in the pre-study, each team should choose one or two representatives who take on the responsibility for assigning the user stories correctly and continuously. The teams are asked to select the team member who is likely to write the most user stories. In most teams, a Business Analyst in the team takes on the

responsibility for the user story assignment. To get started frequent meetings with an enterprise architect and the representative(s) of each team are helpful to assign the first user stories collaboratively to get to know the domains and the approach. In the beginning, these meetings are held every week and are time-boxed to 30 minutes. The meetings are done separately for each of the teams. During that time the participants assign current user stories to subdomains collaboratively. This allows the representatives to get to know the (sub-) domains and their definitions and gives the enterprise architect an impression how easy or difficult the procedure of deciding for the suitable subdomain is. Further, the meeting is used to answer questions that came up when the teams tried to assign user stories. After around four weeks the representatives are able to assign most user stories themselves. Then enterprise architects are mostly responsible to check the user story assignment in terms of quality and completeness. This means they check if the user stories are assigned correctly and question the assignment in the cases in which their assignment might differ. If the teams stop or forget assigning the stories, it is the enterprise architect's responsibility to encourage the teams to proceed with the assignment.

The approach of choosing one representative of the teams to be responsible for the user story assignment proved to be very helpful so that there is one person to be addressed by enterprise architects. However, all team members are able to assign user stories after being taught by the representatives. Some teams also chose to assign the user stories collaboratively from the beginning on in a meeting with the entire team, such as the user story refinement. Therefore, it is beneficial that in the framework it is not formulated exactly when in the process the user stories have to be assigned. The teams can choose when this step fits best in their development process. The only constraint is to have the assignment done latest as soon as a story becomes part of the current sprint. This gives the team flexibility. Therefore, the enterprise architects determine only that the assignment must be made, but not when. In the best case, the teams assign a user story when they create it.

Another important condition is not to slow the teams down because of the additional effort for the pre-study. Therefore, it makes sense if the teams only assign the user stories which are part of the current sprint or in the product backlog. In the case study enterprise architects are also responsible for assigning the already implemented user stories. This is helpful to have more reliable and valid results for each of the teams by getting an overview of the domains of all user stories which were, are and will be implemented. This step is not necessary if the teams start with the user story assignment from the beginning of the development process. Therefore, in the framework it is determined that the teams assign their user stories supported by enterprise architects. While in the beginning of the project more guidance is necessary, it will become less over time. The simplicity of the approach and that the teams are able to do the assignment on their own very quickly is an important result of the pre-study.

The entire evaluation of the pre-study was done in Microsoft Excel, while in the framework the evaluation happens with Iteraplan. The evaluation in Microsoft Excel is useful for quickly gathering and evaluating further information on the assignment of the user stories, such as by whom the assignment was made. This is only evaluated in the pre-

study. Therefore, it was not included in the framework and not in the evaluations that can be made in Iteraplan. The assignment of user stories to domains happens in Jira Software as described in the previous section. To evaluate the assignment it is possible to export all user stories for all three projects including the field "Affected Domain" in a separate file each. After making some adaptations concerning formatting to the exported files (.xls), an Microsoft Excel tool is used to conduct the evaluation. The tool generates an overview of the number of user stories assigned to each domain and subdomain in a Table as well as bar and pie charts. It further shows by whom the user story was assigned and if it could be assigned unambiguously. These results are shown in a Table as well as a bar chart each. Additionally, the file includes an overview - as a list - about all user stories including their key (an identification number), summary, description, sprint, status, domain, subdomain, a field showing if the story was assignable and by whom it was assigned. For some examples, see Table 4.2.

US Key	Summary	Description	Sprint	Status	Domain	Subdomain	Assignment	By?
USA-1	My first User Story	As a ..., I want to... so that...	Sprint 5	done	-		Design US	by EA
USA-2	My second User Story	As a ..., I want to... so that...	Sprint 6	in progress	Domain D	Subdomain D.1	Technical US	by EA
USA-3	My third User Story	As a ..., I want to... so that...	Sprint 7	open	Domain A	Subdomain A.1	unambiguous	in meeting
USA-4	My fourth User Story	As a ..., I want to... so that...		open	Domain B	Subdomain B.2	unambiguous	by team
USA-5	My fifth User Story	As a ..., I want to... so that...	Sprint 5	done	Domain A	Subdomain A.1	ambiguous/ more than one subdomain affected	by EA
USA-6	My sixth User Story	As a ..., I want to... so that...	Sprint 6	in progress	Domain A	none	no suitable subdomain	in meeting

Table 4.2: Example for a list for one team of User Stories to be evaluated in the Excel Evaluation Tool

The field that shows if a user story is assignable provides five different choices:

- unambiguous: selected if it is clear to which subdomain a user story belongs, e.g. US-103 and US-104 belong to subdomain A.1 and B.2 respectively;
- Design user story: selected if no functionality is affected by the user story, but only the design, then the story is assigned to no subdomain e.g. US-101 which could include to move a bottom from the bottom to the top of a page;
- Technical user story: selected if the story is technically necessary to lay the foundation for further functional requirements, e.g. US-102. Technical user stories are opposing to design stories, assigned to a domain including only technical subdomains, here domain D;
- ambiguous/ more than one subdomain affected: selected if a user story affects more than one subdomain or it is not possible to determine to which one it belongs unambiguously, e.g. US-105 most likely belongs to subdomain A.1, but might affect A.2 as well; and

- no suitable subdomain: selected if the domain overview does not include a suitable subdomain yet and it might be necessary to add a new subdomain, e.g. US-106 could belong to domain A, but none of the subdomains might be suitable. Then an additional subdomain A.3 could be added.

As the exact results of the pre-study are sensitive information for the large insurance company, only the results concerning the methodology itself as well as some adapted results are presented. As shown in Figure 4.1 in this case around 55% of the user stories were assigned by an Enterprise Architect, while 18% assigned in the mentioned meetings and 27% by the teams. This will change over time, as at the point of the evaluation all previously implemented user stories were assigned by an Enterprise Architect and all new ones will be assigned by the teams. This information was documented by the involved Enterprise Architect in an Microsoft Excel file.

Assignment	Nr. of US	in %
unambiguous	245	57,65%
Design US	41	9,65%
ambiguous/ more than one subdomain affected	17	4,00%
no suitable subdomain	14	3,29%
technical US	108	25,41%
Sum	425	100,00%

By?	Nr. of US	in %
in meeting	78	18,35%
by team	113	26,59%
by EA	234	55,06%
Sum	425	100,00%

Figure 4.1: Main Results of the pre-study (Tables)

Most of the 425 user stories are assigned unambiguously to one subdomain. This comprises around 58%. However, this low percentage results from the high proportion of design and technical user stories of around 35% which cannot be assigned to a functional domain. It was detected that it makes sense to exclude technical user stories for determining the functional areas for a project. However, it was interesting how much effort the teams have to invest in purely technical issues. Technical issues can be seen - corresponding to SAFe - as a contribution to the architectural runway, as they enable the implementation of business related functions in the future. Further, it is interesting to see how of the total effort the teams invest in design user stories.

The percentage of user stories that fit to more than one subdomain is around 4%. This can be fixed by dividing the respective user stories into several user stories each with focus on the functional requirements of one subdomain.

Around 3% of user stories fit in no subdomain. This is an indicator that the domain model is not entirely complete. As soon as for a potential new subdomain several user stories are found, it makes sense to establish this new subdomain in the overview of domains and subdomains. However, this is only necessary if this domain can be completely delimited without having intersections with other subdomains and if it is likely that more user stories for this subdomain will occur in the future. Based on the results of the pre-study,

one additional subdomain was added. One domain was restructured by renaming the included subdomains and defining their scope differently. The cross-team comparison in Figure 4.2 shows the assignment results for each of the teams. The distribution of the three

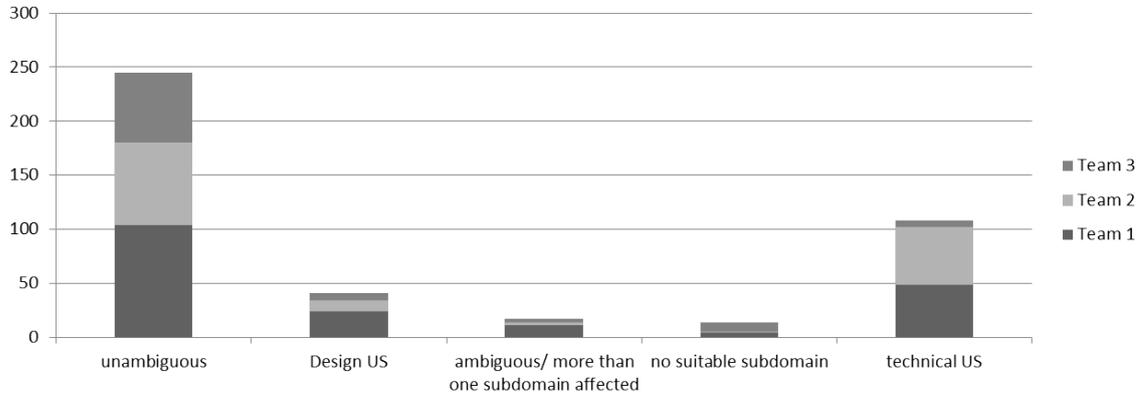


Figure 4.2: Main Results of the pre-study per team (Bar Chart)

teams in the different bars is very similar. However, team 1 seems to have more design user stories than the other teams and team 3 has far less purely technical user stories than the other teams. This mostly depends on the current focus of the projects. Team 1 might have decided on changing some aspects of the user interface at the time, e.g. to improve usability. Summing up, all teams were able to assign the majority of functional user stories unambiguously.

The pre-study results are presented to the teams and discussed with them. Overall the results correspond to their perception of current challenges and overarching functions. It is especially important to present the results to the project managers which are - also according to the framework - responsible for making strategic decisions about the organizational structure. The project managers provided the results of a workshop, which was conducted to identify overarching functions as well as the core functions in the entire project. These results were compared to the pre-study results and the refined by including the numbers resulting from the pre study. Based on these combined results the project managers decided to start one new team with focus on one subdomain - as a test for the framework. The pre-study results help not only to make first decisions based on the approach, but also to get insights in the approach itself. It leads to several findings:

- User stories that relate purely to design issues exist and cannot be assigned to a functional domain. This is detected right in the beginning of the pre-study, so it is possible to include it in the evaluation.
- User stories that solve purely to technical issues and cannot be assigned to a functional domain. This is detected right in the beginning of the pre-study, so it is possible to include it in the evaluation.

- Most of the functional user stories can be assigned unambiguously.
- The approach was easy to understand for the teams and after a few meetings with the teams, they were able to do the assignment on their own.
- Quality assurance by enterprise architects is helpful to detect issues where more discussion can be helpful.
- Comparing the results to the problems the teams reported and overarching functions, they already had identified, the results of the pre-study seem to be reliable.
- Opinions about having precise numbers for making decisions were very positive.
- The results are really used to make strategic decisions about the organizational structure.
- Acceptance of the approach can be supported by explaining the teams the approach thoroughly and presenting the results to them. This also allows to discuss the results with them and compare it to their perception of critical issues.

4.2 Results of the Interviews

In a time-frame of one week, interviews with nine persons in different roles, that are all associated with the agile teams at the large insurance company, are conducted. In the following sections the main results of the interviews are presented. The interviews consist of four main topics. Starting with general questions, questions about strategic DDD, tactical DDD and the development process (in the framework) follow.

4.2.1 General Questions

The first question concerns in which role the interviewee works at the large insurance company. Nine persons with different roles are interviewed. Among the interviewed persons are members of the three agile teams with the roles agile master (AM), business analyst (BA1 and BA2) and product owner (PO). Further, the two projects managers (PM1 and PM2) of the respective project are interviewed. Next to them, a department head in operational organization (DH) sales takes part in the interviews. Further, a domain architect in the sales domain (DA) and a lead enterprise architect (EA) are interviewed. Table 4.3 gives an overview of the roles of all interview partners as well as their respective alias which will be used in the following to reference their answers. The table also shows if the interview partners took part in the user story assignment (strategic DDD) and in event storming workshops (tactical DDD).

Secondly, the interviewees are asked how long they have been part of an agile team or how long they work in cooperation with agile teams. AM, BA1, BA2 and PO work in their

Nr.	Role	Alias	User Story Assignment	Event Storming
1	Project Manager	PM1		
2	Agile Master	AM	X	X
3	Business Analyst	BA1	X	
4	Business Analyst	BA2	X	X
5	Product Owner	PO	X	X
6	Department Head of Sales Processes and Applications (Operational Organization)	DH		
7	Lead Enterprise Architect	EA		
8	Project Manager	PM2		
9	Domain Architect (Sales Domain)	DA		X

Table 4.3: Interview partners and their participation in user story assignment and Event Storming workshops

respective agile teams since one and a half year. Of them only PO has worked in an agile team before. Both project managers (PM1 and PM2) work with agile teams since one and a half year. EA has contact with agile teams since 2012, however he is currently not working with the agile teams on the project which is part of the case study. DH has experience with agile teams of around ten years and works with the agile teams which are part of the case study since their start one and a half year ago. DA has around six years of experience with agile teams including the ones considered in the case study.

The third question addresses the major challenges the interviewees experienced in agile development with more than one team. Many mention that coordination and communication between the teams is one of the biggest challenges (BA1, BA2, EA, PO and PM2). While some say there is too less exchange between the teams, some also mention that it is a problem that it is not actually defined how and when the teams should exchange knowledge. Further, it is not clearly defined how the teams work together if they require something from another team (PM2). Additionally, the reuse of overarching functionality can be a challenge. While the teams oppose having to reuse code from other teams as they have varying requirements, all agree that there must be a way to access functionality implemented by other teams e.g. via APIs (BA1 and BA2). AM and PM2 also stress the problem, that each team has its own goals which it has to reach without having an overarching goal all teams need to achieve. Therefore, there often is no time for knowledge exchange or focus on overarching functionality and each team tries to find an own solution. DA adds that this problematic should be addressed by adapting the management of agile teams including goal setting as well as budget planning. The challenges include further that each team is working very independently from each other and currently there has

been not defined how to scale with even more agile teams (DA). In this context, the question arises in which architectural and technical questions the teams decide themselves and when overarching decisions need to be taken as the applications they develop later have to fit in the overall application landscape (PM2, DH). EA further raises the question, if decentralized agile teams really are suitable for developing very large and complex applications. DH stresses the importance of architectural considerations before starting a development process. In DH's opinion, the (sub-)domains the teams work in must be clear beforehand to determine when they need something from other teams.

4.2.2 Strategic DDD

The first question concerning strategic DDD (question 4) is if the interviewee participated in the user story assignment to the subdomains. Of the nine interview partners, four took part in the assignment: AM, BA1, BA2 and PO (see also Table 4.3). Question 5 is only asked to them. This question concerns the time at which they assign the user stories to a subdomain. In the best case, this would be done right after creating a user story. BA1, BA2 and PO assign the user stories right after creating them if it is not forgotten. AM assigns the stories later in the process as soon as a user stories becomes part of a sprint and really gets implemented, because often user stories become unnecessary and are closed.

Additionally, the interview partners are asked in question six if they already have profited from the evaluations of the user story assignment and if yes how. The agile team members (AM, BA1, BA2 and PO) answer - as expected - that the teams couldn't make use of this artifact so far. BA2 says that the evaluation might be useful for the Product Owners. PO added that the evaluation proved correct the assumptions about the overarching (sub-)domains they work in. DH, PM1 and PM2 confirm this and add that the evaluation delivered data about the core focus of each team and the overarching functions that might require some more discussion and ultimately also action. They see it as a valid basis for their decisions about team structure and responsibilities. According to DA, the results were used to create a decision memo for the strategic enterprise architecture management board and for project managers. EA adds that the evaluation if presented on a dashboard gives decision makers a nice overview of those subdomains that require action. DH further states that this considerations should have been made before starting the development process.

Starting with question seven the most following questions include a statement which the interviewees are asked to rate on a Likert scale from 1 - I strongly disagree to 5 - I strongly agree (2 - I disagree, 3 - I neither agree nor disagree, 4 - I agree). The results are shown in bar charts with two different colors: blue for members of agile teams (AM, BA1, BA2 and PO) and grey for decision makers, project managers (PMs) and architects (PM1, PM2, DH, EA, DA).

Statement seven is if it makes sense to assign the user stories to subdomains further and to evaluate them continuously. This statement receives a rather high agreement with an average of 4,1. However, the members of agile teams rate the statement lower than the other interviewees. This can be seen in Figure 4.3. A reason for that is that the teams haven't

really profited from the results so far. However, AM says that making the benefits of the assignment more clear to the teams could help them to use the results. PO considers it beneficial to identify overarching subdomains continuously to identify need for action. DA says that he has seen some cases in which the results were used, however it is necessary to find some additional use case and possibilities to scale the approach in the future. The further reasoning is similar to question six. In general, reactions to this statement are rather positive.

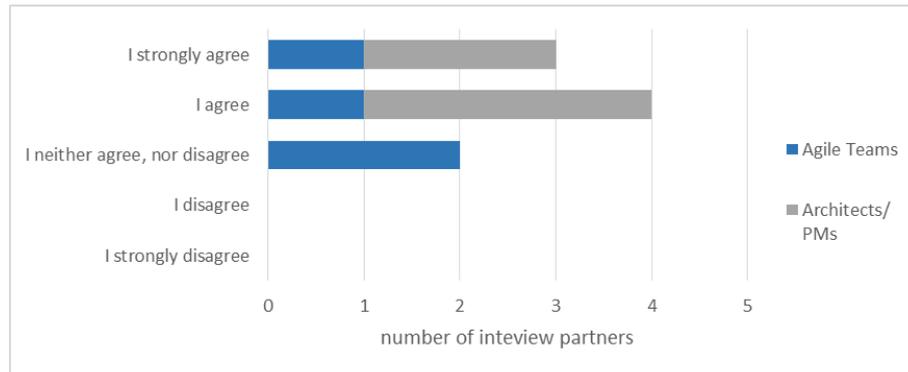


Figure 4.3: Question 7: It is beneficial to further assign user stories and to evaluate the assignment continuously (from 1 - I strongly disagree to 5 - I strongly agree).

Question eight includes a statement concerning the presentation of the results of the user story assignment to all stakeholders in form of charts on a Wiki page. Here agreement is even higher with an average of 4,4. There aren't really differences in the ratings by agile team members and the other interview partners (see Figure4.4).

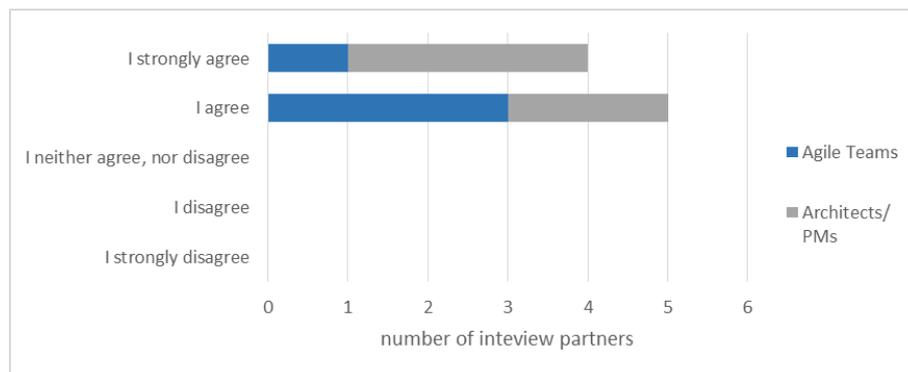


Figure 4.4: Question 8: It is beneficial to provide the continuous results of the user story assignment to all involved persons on a Wiki page (from 1 - I strongly disagree to 5 - I strongly agree).

All consider the Wiki as the currently most suitable tool to provide the results as this increases transparency (EA), might be easy to find (PM2) and teams use the wiki (PM1). DA considers it as overall helpful, but is not sure if the teams really will use it. DA sees having the results in Jira more as a useful tool for architects to detect areas they should focus on in an easy ways. Then architects can address the detected issues easily with the teams by referring to these easily accessible results. After that architects and agile teams can address the issues collaboratively.

4.2.3 Tactical DDD

Question nine which is the first question concerning tactical DDD addresses its most central artifact: the domain model. The statement is that it is beneficial that each team creates and evolves its own domain model. The agreement in this question is on average 4,8 (see also 4.5). The model helps the teams to reach a common understanding of the business logic in its domain and serves as a shared language for all team members - developers as well as business experts (DA, PO, EA). AM sees the domain model as very helpful to detect which functionality could be added next and as tool to discuss how the logic changes. DA adds that in the future this domain model also has to be found in the code. Developers should base their code on the domain model which is a representation of the business logic. All respondents agree strongly with the statement in question nine except PM1 who notes that it might make more sense in the future if the teams are structured based on subdomains. Then each team would work on one subdomain and establish a domain model of this subdomain as defined by DDD. However, PM1 adds that it takes still a long time to structure teams in that way and that it is questionable if it is realistic to do so.

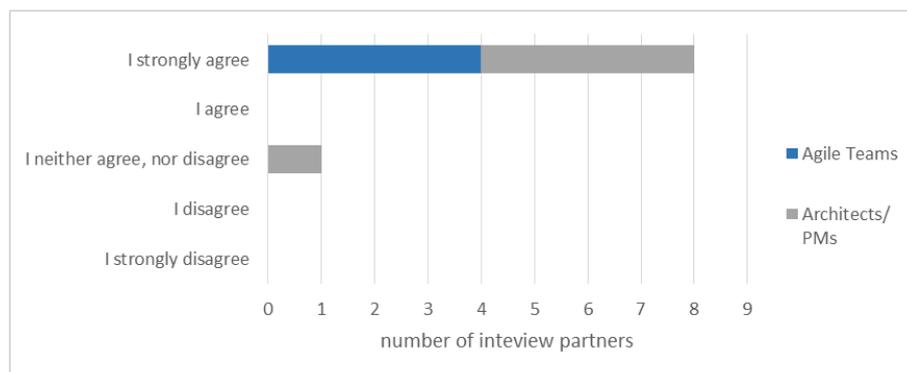


Figure 4.5: Question 9: It is beneficial if each teams creates and evolves its own domain model (from 1 - I strongly disagree to 5 - I strongly agree).

Question ten is if the respondents took part in an event storming workshop. Four of

nine interview partners participated in one or more workshops: AM, BA2, PO and DA (see Table 4.3). Question 11 is only asked to the participants and is in which role the persons participated. AM, BA2 and PO took part as representatives of their respective agile team in their assigned role. DA was a moderator in two of the event storming workshops so far. Question twelve is only posed to the participants of the workshops. This question concerns the use of the domain model in the teams. However, the interviews were conducted after event storming workshops with only parts of the teams and it was decided to present the results later to the other team members. Therefore, AM, BA2 and PO answer that they didn't use it so far, but they are very positive to use it in the future for naming conventions or language, determining the scope of services (BA2) and to refine user stories and the according events as well as for scoping and planning in the beginning of the development process and as a basis for discussion about unclear topics (PO).

Question 13 is if event storming as a method is beneficial to establish a first draft of the domain model and to refine it. The statement received a very high agreement with an average of 4,8 (see also Figure4.6). In general the team members who participated in the workshops are really convinced that the method is very helpful, but also all others see it as a helpful and method. Advantages of the method include that it is very simple and understandable (PO, AM), that it creates a good atmosphere for open discussions (BA2, EA), that its focus is on business events and logic (BA2) and helps to find aggregates and (sub-)domains boundaries (PM1). DA adds that it is a good thing to focus on the business logic before addressing technical issues.

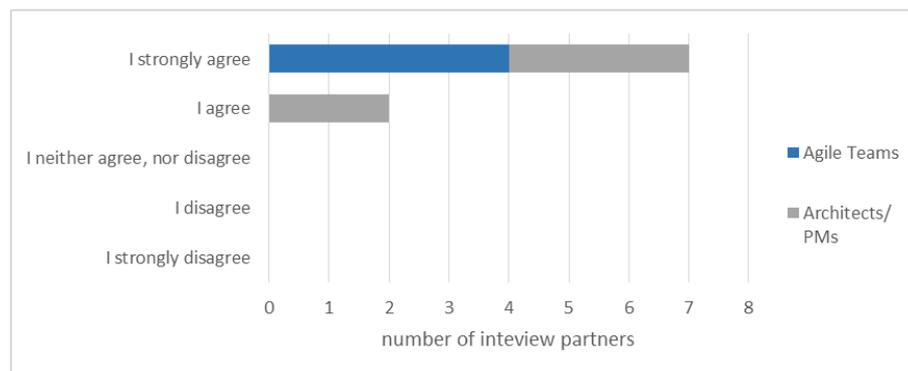


Figure 4.6: Question 13: It is beneficial to use the event storming method to create a (first draft of a) domain model (from 1 - I strongly disagree to 5 - I strongly agree).

Question 14, which is the last concerning tactical DDD, is if it is beneficial to compare the domain models if two teams have dependencies on each other. This statement receives an average agreement of 4,6 (see also Figure4.7. Reasoning includes that it helps to determine the boundaries of each team's work, which would be in the best case also the boundary of the subdomains (BA2). Therefore, each team needs its own domain model before a com-

parison can be made. Then it might help to reach a common understanding of the business logic where interfaces between the applications are required. Possibly this could also help to define a published language (DH). Further, if this is done timely problems, which otherwise might occur later caused by dependencies, can be detected early (EA and PM2). However, this approach can increase the need for cross-team coordination and communication (EA). If this is wished the interdependency could increase (BA1). Further, this might not be desired by the teams as they are supposed to reach their own goals and therefore interest in common goals and overall architecture might not be given (AM). DA sees it as the architects' responsibility to compare domain models and to detect challenges that might occur. An architect then could contact the affected teams and bring them together to discuss and decide about the detected issues.

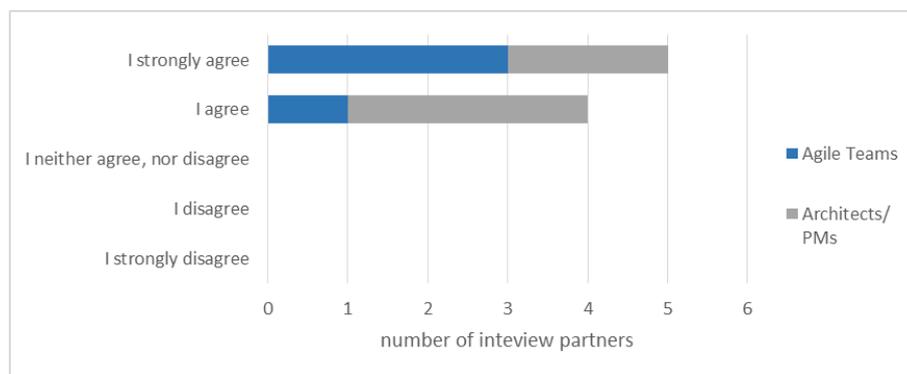


Figure 4.7: Question 14: It is beneficial if teams with dependencies compare their domain models in a workshops (from 1 - I strongly disagree to 5 - I strongly agree).

4.2.4 Development Process

The next questions address the potential development process with three example teams as depicted in the defined framework. Interviewees were asked about some of the components included in the framework.

Question 15 is how the interviewees estimate the effort for the teams to include the strategic and tactical DDD component into their development process. The interview partners are requested to make an estimation on a scale from 1- very low to 5-very high. Here the average estimation is 3,1 meaning a medium high effort for the teams. Surprisingly, all interviewees choose a number between two and four and agile teams don't estimate the effort higher than the other participants. This is also shown in Figure4.8. Reasoning includes that effort is not that high and that the teams especially can profit from the tactical component directly and in the long run also from the strategic component (PM1, PM and EA). Some stress it is worth the effort or that it has to be done anyway (DH). However, PO mentions that now the effort is small, but that it might increase in the future when a lot

of action has to be taken. DA adds that also a mind change might be necessary, from addressing challenges from a more technical to addressing them with focus on the business logic.

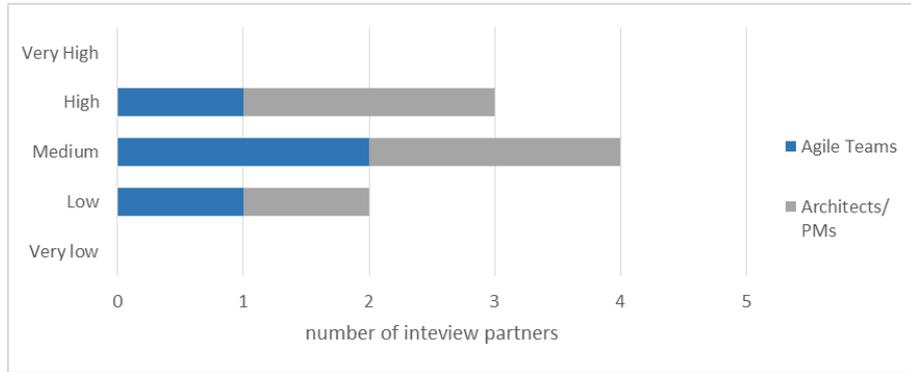


Figure 4.8: Question 15: How do estimate the effort for the teams to integrate the strategic and tactical DDD component in the development process (from 1 - very low to 5 - very high)?

Question 16 asks from the interviewees to judge if it makes sense to synchronize the sprints of teams with dependencies (same sprint length, start and end). Here the opinions vary in dependently from the role of the interviewees (see also Figure4.9).

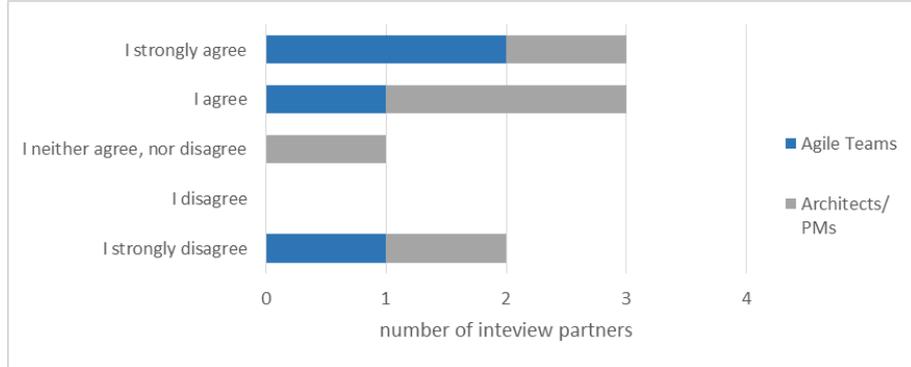


Figure 4.9: Question 16: It is beneficial to synchronize the sprints of teams with dependencies (same sprint length, start and end) (from 1 - I strongly disagree to 5 - I strongly agree).

The ones that agree to the statement argue that this helps to manage dependencies and might be easy to operationalize (PM1). Further, two of the considered teams were one team before being divided. In this case, there are still some dependencies and then it e.g.

makes sense to synchronize the sprints (BA1). AM also thinks it makes sense and adds that in this context it might help to have very short sprints, maybe around one week, to react quickly. EA says if homogeneity and complexity of the topics the teams work on are similar it might work. BA2 argues that actually no dependencies should exist and every team is responsible for its own API and that in this case every team should decide about its sprint length. If continuous delivery and integration is practiced, it also is not necessary (DA, PM2). An important addition by DA is that synchronizing sprints are only useful if the teams all work on one common product. Currently, the organization is that each teams works on an own product and has its own goals, however they all belong to one project with the goal to have one common sales platform including all products.

Question 17 is divided into questions 17a and 17b. Question 17a is if the interviewees think it makes sense if teams with dependencies share a single product backlog. Here the average agreement is around 3,2. However, the opinions vary which can be also seen in Figure 4.10. The ones that agree argue that it might help to keep whole product/ project focus (EA) and to increase transparency (PM1). PM2 argues there should not be dependencies and then it makes no sense to share a backlog. DH also says a single backlog does not make sense as long as each team knows its scope and can manage the dependencies that exist. Further, BA1 explains that each teams has a high number of different user stories and with one backlog it would be very confusing. DA adds that the decision for one or multiple backlogs depends on how the teams are structured and if they work on one common product or different ones.

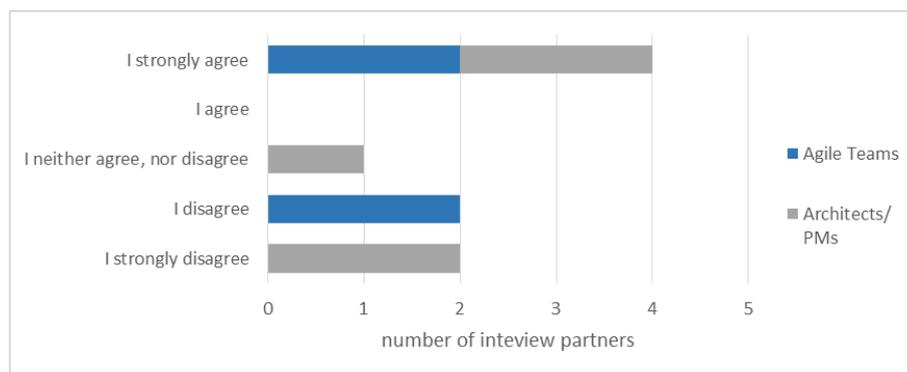


Figure 4.10: Question 17a: It is beneficial if teams with dependencies share a single backlog (from 1 - I strongly disagree to 5 - I strongly agree).

Question 17b is teams with dependencies in one project should have a common product owner. Here the opinions also differ and the average agreement was around 2,7 (see also Figure 4.11). When asking this question, it comes apparent that different persons have different opinion on what a product owner has to do in this case. If the product owner focuses on prioritizing and focusing on an overarching goal, some say it makes sense to

have only one product owner. With many teams in one project it also helps if someone with an overview of all teams prioritizes the user stories (AM, EA). PO argues that the agile teams need to be experienced to make it work with one overarching product owner as they have to refine and organize their work themselves in this case. If teams are not as experienced they might need a product owner each, plus an additional product owner for the overall prioritization of user stories (PO). Further, some argue that handling an overarching backlog is too much work for a single product owner (DH, PM2). DA says that for him the decisions for a single backlog and a single product owner are connected, meaning that for one backlog there should be always one responsible product owner.

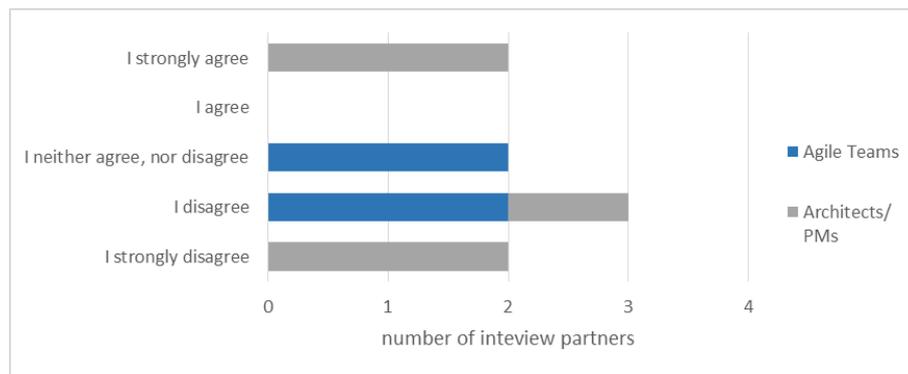


Figure 4.11: Question 17b: It is beneficial if teams with dependencies are managed by a single product owner (from 1 - I strongly disagree to 5 - I strongly agree).

Question 18 includes a very broad statement: It is beneficial if teams use the defined artifacts, such as domain model and user story assignment results, in the overarching events (sprint planning, refinement, review, retrospective). The statement receives an overall agreement of 4,4. However, AM and BA1 don't answer the question, as they don't try applying the artifacts so far (see also Figure 4.12). PM1 says the artifacts, can be useful for scoping the next MVP and help to determine which functional area the implementation of an user story affects. EA explains that it is necessary, but only if especially the strategic DDD component is given and applied. PM2 agrees and argues that it is not a high effort, but that it really has to be applied to generate business value. DH adds that the domain model is supposed to serve as the basis for a shared language in the teams and therefore the domain model is utilized in that way in every meeting. Others explain that it depends on what you want to reach in a meeting if it makes sense to use a certain artifact (BA1, DA). DA thinks that the domain model can be very helpful in the refinement as a shared language and very good for MVP scoping as well as for developers for the implementation of certain user stories.

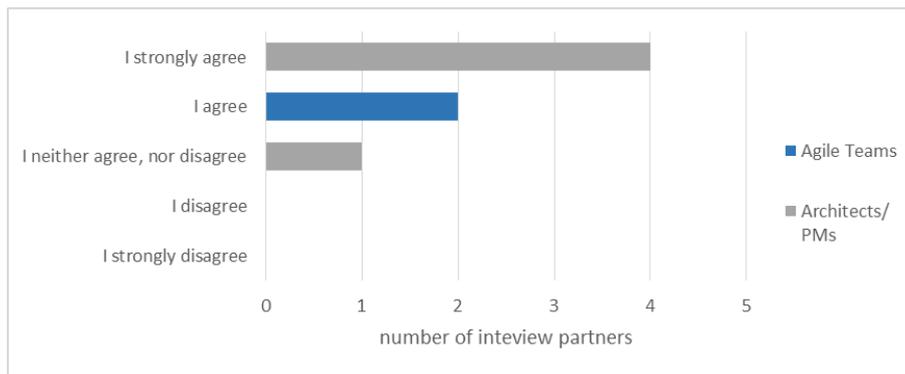


Figure 4.12: Question 18: It is beneficial if teams use the defined artifacts, such as domain model and user story assignment results, in the overarching events (from 1 - I strongly disagree to 5 - I strongly agree).

4.2.5 Final Questions

Question 19 includes the statement: It is beneficial if architects support the agile teams as described by the framework. This statement receives full agreement (average 5,0) by all interview partners (see also Figure 4.13).

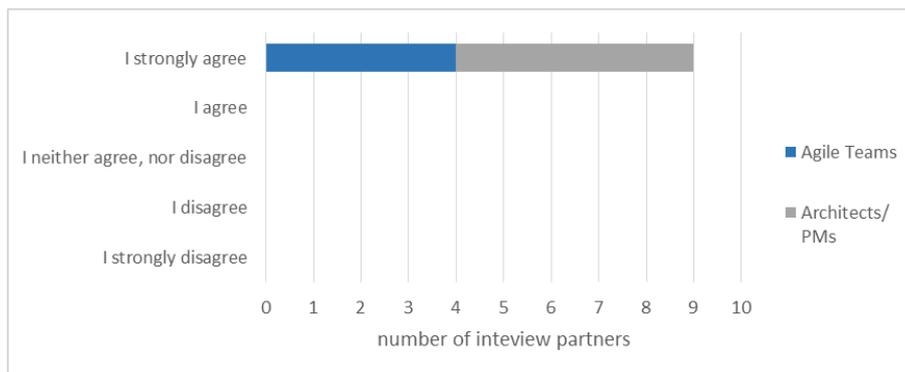


Figure 4.13: Question 19: It is beneficial if architects support the agile teams as described by the framework (from 1 - I strongly disagree to 5 - I strongly agree).

Some say that architects should coach the teams concerning new methodologies that can be helpful for them as well for overarching considerations, such as DDD (PM1, AM BA2). Architects are supposed to have an overview of all involved teams and applications and therefore should provide input for overarching issues, such as team structure depending on domains (DH, PO, BA1, BA2). EA sees also the necessity to establish some overarching architectural standards in the agile teams. Further, architects are seen as moderators

for inter-team issues and in event storming workshops (PO). To get started with DDD the teams need someone - an architect - to establish the approach initially and keep the responsibility for all issues connected to it (PM1). DH sees the architects explicitly not in a supporting role, but says that it is their task to make especially the strategic considerations about team structure and in which domains the teams work before the development process gets started. In his opinion, the architect has a very central role connected to the agile teams. DA agrees that architects need to cooperate with project managers to make strategic and architectural decisions concerning multiple teams.

Finally, the interviewees were asked if they have something to add concerning the framework and its approach in the agile teams or if they have any other additions concerning the interviews. PM1 stresses that the operationalization of the framework has to be continued and scaling the framework to more than the current three teams is needed. BA1 adds that coordination and communication between the teams is always difficult and takes a lot of time. However, in his opinion overarching components the teams could access would be beneficial. DH commented on the framework saying that the division in the three components makes sense. However, he adds that the results of the strategic DDD pre-study were not surprising, but helped the decision makers. Further, the learnings need to be discussed and the process changed accordingly (if necessary).

5 Discussion

In this chapter, the findings connected to the challenges - defined in Table 3.1 - as well as further key findings are explained. This thesis connects agile scaling frameworks and DDD to address challenges which arise in large organizations with several agile teams. Additionally, some limitations are discussed.

Addressing the defined challenges In this thesis, strategic DDD is used to address the challenges concerning organizational structure and team responsibilities (C5 and C6). The goal is to structure teams according to the subdomains they work in. This means restructuring already established teams based on the user story assignment - if necessary. Additionally, new teams should be structured around the subdomains they mostly work in. However, this would require some additional upfront effort. Structuring teams based on subdomains helps to define clear boundaries around a team's responsibility and, according to DDD, helps to reduce dependencies between teams (C1). Further, strategic DDD would help the current teams to focus on their core domains and new teams with focus on subdomains including mostly overarching function could be established (C2). This would overcome the current challenge that each of the teams develops its own overarching functions e.g for managing documents or roles and legitimations.

Tactical DDD addresses challenge C11 by using domain modeling to reach a shared understanding of the business logic within each of the teams. Further, domain models help to define an ubiquitous language for each of the teams.

Agile scaling, similar to LeSS, as in the framework for (enterprise) architecture in agile teams provides a process for scaling the Scrum method to more than one team (C7) for cross-team coordination (C3) and inter-team communication (C4). All overall meetings as well as sharing a product backlog and a product owner serve this purpose. This also addresses the challenge of maintaining the whole product focus in all teams (C8). However, in the case study the interview participants had different opinions with some saying the three teams work on one product and others saying on three completely different ones. This leads to a lack of common goals.

By introducing DDD and defining the role of enterprise architects in the framework the lack of architectural guidance (C8) and the lacking definition of the enterprise architect role (C9) are addressed. In the framework, architects found their position in agile development projects as guides concerning new methodologies, as moderators and facilitators. Cultural change towards a more lean and agile mindset in the company (C12) can only happen over time, but using agile methodologies, coaching and implementing DDD can

be a step towards such a mind change.

Key Findings After working independently from all architectural governance, the agile teams and project managers realized that without any form of architectural guidance large agile projects can hardly be successful. Therefore, one of the key findings is that agile teams, as soon as there are several of them on a project, need to be supported by enterprise architects having an overview of the teams and the applications they develop. Many challenges arise which cannot be addressed by single teams, but need to be addressed with overarching methods driven by overarching roles within the organization.

Especially, combining agile scaling and DDD can address various challenges. While scaling frameworks support cross-team coordination and communication, they lack detailed advice on how to do architecting in large scale agile projects. Therefore, the combination of agile scaling and DDD can be beneficial.

DDD provides basic concepts for architecture that can be beneficial not only to the agile teams, but the project overall. Architectural activities in agile projects earlier were not accepted by agile teams who wanted to work independently. However, if architects are capable of providing apparent value to the agile teams, they appreciate architectural support. The same applies for project managers and other decision makers.

To be able to demonstrate value quickly to both decision makers and agile teams, we recommend starting with both strategic and tactical DDD at the same time. Decision makers will profit soon from the strategic DDD, while agile teams profit mostly from the tactical component in the first place.

Further, the framework shows how to combine agile scaling and DDD in a lightweight method. Before, it was only recommended to combine the approaches without describing how to do so.

Key findings about the role of the enterprise architect include that it can be helpful if this role is part of a large project to have an overview of all teams and their subdomains to detect where action is required. Further, stakeholders involved in the project appreciate that architects not only coach the teams concerning new methods, such as DDD, but also support them to actually apply them and get value out of them quickly. This also includes being an objective moderator in event storming workshops, such as depicted in the framework. It can be beneficial for architects to work with the teams at their location continuously to support the DDD process over time.

For scaling, the agile teams and project managers must be on the same page if they work all on a single product or on different ones. This seems to be a challenge in the case study. Here the teams say they work on their application each with completely different requirements than the other ones, while project managers consider the applications developed by the three teams as a single product. However, before starting to scale one should discuss this question in detail. Based on this the goals and budget for the teams need to be set. The assumption is that if teams work on a single product, there must be some overarching goals all teams contribute to and based on which the teams' success is measured. Then

it might help the teams to also focus on overarching goals, e.g. concerning overarching functions, for which they now in the current situation don't have time without failing to reach their own team goals.

Limitations There are several limiting factors for the internal validity and generalizability of the case study results. First, the case study is conducted only in a limited timeframe, so the benefits and drawbacks of the framework could not be evaluated in the long run. There is a decision about establishing a new team with focus on one overarching subdomain. However, this team was not established yet, so it is not possible to evaluate the consequences of this decision.

After having evaluated the framework, a recommendation would be to start the DDD processes based on the framework already some time before the teams start developing with event storming workshops. After the evaluation all stakeholders see it as a very good method, but not all use cases for doing event storming could be tested. This was not possible in the case study as the teams already had implemented significant functions of their applications. Based on the evaluation, we would recommend conducting a case study which establishes DDD from the beginning on.

Further, the development process is not established as depicted in the framework so far and is only a possible target process. However, the focus in this thesis is on the strategic and tactical DDD process which is tested with the teams. However, the development process as defined in the framework could be tested in the future.

Only three teams participate in the case study, while the other agile teams on the project do not participate so far. To test the entire framework, we would recommend that all teams on the project take part in the DDD process for improved and more complete results.

The case study and its evaluation in this thesis are only done in one company - the large insurance company. Repeating the approach in other organizations might yield different results and recommendations for adaptations. Other companies have different prerequisites, such as experience with agile methods and scaling, a different culture or organizational structure. Further, experience of architects and their acceptance in agile teams may differ. The framework for (enterprise) architecture in agile teams is not like a "cookbook" giving a step-by-step advise about scaling and establishing DDD. The framework is described and evaluated as it was conducted in the large insurance company and requires adaptations to fit another company and its goals.

The DDD approach, according to the book by Evans, also means to change how developers interact in their daily work [30]. DDD especially describes how projects proceed within their bounded contexts. DDD gives a more detailed approach to developing complex software on the development level. The defined framework for (enterprise) architecture in agile teams only deals with the first steps to take for establishing DDD. The process might need to be adapted over time and further DDD concepts, such as context mapping on the strategic level and further building blocks on the tactical level, can be incorporated in the framework for (enterprise) architecture in agile teams.

One major drawback of the framework is the term of the domain model. The definition of the domain model in DDD is completely different from the one enterprise architects know. Therefore, teaching suitable terms from the beginning on is absolutely necessary to create a shared understanding of this term. Another option would be to adapt the terms fitting to the language in the teams.

6 Conclusion

Finally, in this chapter the three research questions are answered in the summary before an outlook on further research is given and practical applications of the framework for (enterprise) architecture in agile teams are explained.

6.1 Summary

In the following, the findings answering the three research questions are summarized.

RQ1. What is the role of architecture in scaled agile organizations? The three agile scaling frameworks have very different views on the role of architecture in large scale agile development. Nexus does not really address the topic of architecture. Therefore, one can assume that - as in Scrum - architecture evolves through programming and the teams make all architectural decisions themselves. However, the integration team is responsible for teaching architectural standards of the company to the teams. In LeSS, a lot more advice on how the teams evolve architecture in the best way is given. It is said that experienced programmers in the team take on the role of an IT architect. Further, agile modeling and architecture documentation workshops are suggested. In LeSS, only architects on the team level are considered necessary and the role of the enterprise architect is not defined. However, the importance of architectural decisions is acknowledged and it is suggested to use DDD connected to agile (domain) modeling. SAFe is the only framework addressing, next to emerging design, additionally intentional architecture which comprises architectural decisions that affect more than one team and that are out of scope for team decision making. Here the role of the enterprise architect is defined. The enterprise architect aligns the teams on all levels to follow a common technical vision and gives a strategic direction to all teams. SAFe also suggests to use DDD for domain modeling. Summing up, the main takeaway is that DDD is a suggested method to develop architecture without contradicting agile values. It does not require extensive upfront design, but especially in strategic design a business or enterprise architect is needed to keep an overview of bounded contexts and their dependencies. This supports structuring the teams. With clear bounded contexts and reduced dependencies, agile teams can make technical and architectural decisions within their contexts themselves.

RQ2. How can Domain-Driven Design be adopted in a large organization with several agile development teams? The adoption of DDD takes time and should be integrated step-by-step into the development process of agile teams. There should be a specialist concerning the methodology to provide guidance to the teams and to support them throughout the process. In this case study, enterprise architects take on this role. Before getting started the business value of DDD should be explained to foster initial acceptance. Throughout the entire adoption process the value of DDD should be demonstrated to the agile teams and decision makers.

To integrate DDD in the development process one can differentiate between strategic and tactical DDD as already suggested by Evans (2003) [30]. The results of the strategic DDD process are in the first place only used by project managers to make decisions about team structure and responsibilities. The teams themselves might profit later from being able to focus on their core subdomains as well as from reduced dependencies. However, strategic DDD can only work with the input from the agile teams.

Tactical DDD is a bottom-up approach that can be started in the agile teams. However, the teams should focus on their sprint goals and someone from outside the team needs to provide methodological guidance - in this case the enterprise architects. Domain modeling can be a helpful tool to reach a shared understanding of the business logic and an ubiquitous language within a team. The agile teams in the case study regard the domain models, including event storming as the method to get started with domain models, as beneficial. In the case study, the adoption of tactical DDD was started a considerable amount of time after the teams started their development process. In the future, it would be beneficial to conduct event storming workshops before the development starts for planning, scoping and as a comprehensive representation of the business logic. After having understood the business logic, the teams can make technical considerations.

These are the first steps for the adoption of DDD in several agile teams. Throughout the process the teams should be structured according to subdomains. Then it also would make sense to include context mapping into the strategic DDD process to model dependencies between the different subdomains the teams work in. On the tactical level, the teams could further evolve their domain models including additional building blocks as defined by DDD.

RQ3. Which roles, processes, artifacts and tools are required for scaled Domain-Driven Design? The central roles, processes, artifacts and tools are depicted in the defined framework for (enterprise) architecture in agile teams. In the framework, no additional roles are added. The different Scrum roles already existed in the large insurance company, as well as the project managers and enterprise architects. This might also be the case in other companies. However, in the process of adopting DDD the roles have some further responsibilities. The teams provide input for the strategic DDD process to reach those results of the user story assignment based on which project managers and product owners decide about team structure and responsibilities. Further, the teams are advised to evolve domain

models and use them throughout their development process. Especially, the importance of architects in the process increases. Enterprise architects provide methodological guidance to the teams, evaluate the user story assignment, support project managers and product owners in their strategic decision making. Furthermore, they should keep an overview of overarching functions and serve as moderators in event storming workshops. They further might take part in some overall meetings in the development process, if the teams wish to adapt their domain model or want to discuss the results of the user story assignment.

The development process in this framework is largely based on LeSS plus the strategic and tactical DDD the teams contribute to. Major artifacts are - as in Scrum - the product backlog and user stories. However, now the overview of subdomains and the results of the user story assignment become more central also to agile teams. Essential additional artifacts - especially to the teams - are the domain models as well as key use cases and architectural documentation according to arc42.

The teams are advised to keep the tools that they already used and architects integrate them into the DDD processes. Jira and Confluence are very central to the teams also in the framework. Further, architects use their enterprise architecture management tool to document and evaluate the user story assignment.

The framework for (enterprise) architecture in agile teams provides the essential roles, processes, artifacts and tools to establish DDD and therefore a lightweight approach to architecture in scaled agile organizations with several agile teams. However, this might not be complete and so far was only tested in one company in a limited timeframe. Therefore, an outlook on future research is given.

6.2 Outlook

In the future, within the large insurance company the framework and especially its DDD process will be further operationalized. This includes that one enterprise architect will be at least one to two days at the co-location of the agile teams to support the implementation of the DDD process and to support the teams in all overarching and architectural questions. Until now, only three teams on the project were involved with the DDD process. In the future, all agile teams on the project should conduct the user story assignment and evolve a domain model. Further, all new teams are taught the methodology and should get started with DDD right in the beginning. Especially, event storming and domain modeling can be beneficial to them in the scoping and planning of their development process before they actually have implemented something.

Step-by-step in the large insurance company the agile teams could be structured based on subdomains to explore the benefits and drawbacks of the method in the long run.

Additionally, the strategic DDD process addresses the allocation of user stories to bounded contexts which is a crucial step towards DDD. Strategic DDD also suggests context mapping to explore the dependencies between bounded contexts. In the future, this could also be helpful for an overarching view and therefore might be integrated in the framework.

Connecting the strategic and tactical DDD process to explore aggregates and bounded contexts might be helpful as well.

The key use cases and arc42 documentation were only shortly mentioned in the framework, but not really applied and tested so far. These concepts could be further explored and tested increasing the importance of architecture in agile development projects.

Further, one should decide how to scale the framework and the development process. This can be done using an approach similar to LeSS as suggested, but should be decided and tested in the future.

Research concerning how DDD can be established in large and very complex organizations might deliver some interesting use cases that help organizations to adopt DDD. The generalizability of the framework could be explored by trying to apply it in other organizations and to adapt it.

Research concerning the role of architecture in other agile scaling frameworks could be analyzed. Additionally, some research about the implementation in large and established organizations might deliver interesting results on this topic which was not extensively explored so far. The role of architects in agile development projects could be analyzed not necessarily starting from agile scaling frameworks.

For the enterprise architecture management discipline it might be interesting to analyze how this approach and these responsibilities of enterprise architects can be connected to agile enterprise architecture management. To sum up, based on this work a lot of research topics can be explored and applied practically, as many companies face similar challenges.

Bibliography

- [1] Pekka Abrahamsson, Muhammad Ali Babar, and Philippe Kruchten. Agility and architecture: Can they coexist? *IEEE Software*, 27(2), 2010.
- [2] Blue Agility. Scaling agile: Less vs. safe. <https://blue-agility.com/scaling-agile-less-vs-safe/>, 2016.
- [3] Agile Alliance. Agile 101 - manifesto for agile software development. <https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/>, 2017.
- [4] Scrum Alliance. Large scale scrum - more with less. <https://www.scrumalliance.org/why-scrum/agile-atlas/agile-atlas-common-practices/planning/december-2013/large-scale-scrum-more-with-less>, 2013.
- [5] Scrum Alliance. The 2017 state of scrum report, 2017.
- [6] Scrum Alliance. Scrum guide. <https://www.scrumalliance.org/why-scrum/scrums-guide>, 2017.
- [7] Scrum Alliance. Scrum values. <https://www.scrumalliance.org/why-scrum/core-scrum-values-roles>, 2017.
- [8] 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.
- [9] Samuil Angelov, Marcel Meesters, and Matthias Galster. Architects in scrum: What challenges do they face? In *Software Architecture: 10th European Conference, ECSA 2016, Copenhagen, Denmark, November 28–December 2, 2016, Proceedings 10*, pages 229–237. Springer, 2016.
- [10] Atlassian. Confluence. <https://www.atlassian.com/software/confluence>, 2017.
- [11] Atlassian. Jira software. <https://www.atlassian.com/software/jira>, 2017.

- [12] Muhammad Ali Babar. An exploratory study of architectural practices and challenges in using agile software development approaches. In *Software Architecture, 2009 and European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on*, pages 81–90. IEEE, 2009.
- [13] Stefan Bente, Uwe Bombosch, and Shailendra Langade. *Collaborative enterprise architecture: enriching EA with lean, agile, and enterprise 2.0 practices*. Newnes, 2012.
- [14] Charles Bradley. Comparing and contrasting the 3 major agile scaling frameworks (safe, less, nexus). <http://www.scrumcrazy.com/Presentations>, 2016.
- [15] Alberto Brandolini. Introducing event storming. <http://ziobrando.blogspot.de/2013/11/introducing-event-storming.html>, 2013.
- [16] Scrum Breakfast. Scaling scrum: Safe, dad, or less? <http://www.scrum-breakfast.com/2013/10/scaling-scrum-safe-dad-or-less.html>, 2013.
- [17] Scrum Breakfast. Three things to like about safe. <http://www.scrum-breakfast.com/2014/05/three-things-to-like-about-safe.html>, 2014.
- [18] Nanette Brown, Robert Nord, and Ipek Ozkaya. Enabling agility through architecture. Technical report, DTIC Document, 2010.
- [19] Mike Cohn. *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.
- [20] The LeSS Company. Architecture and design. <https://less.works/less/technical-excellence/architecture-design.html>, 2017.
- [21] The LeSS Company. Introduction to less. <https://less.works/less/framework/introduction.html>, 2017.
- [22] The LeSS Company. Organizational structure. <https://less.works/less/structure/organizational-structure.html>, 2017.
- [23] The LeSS Company. Product owner. <https://less.works/less/framework/product-owner.html>, 2017.
- [24] The LeSS Company. Role of manager. <https://less.works/less/management/role-of-manager.html>, 2017.
- [25] The LeSS Company. Scrum master. <https://less.works/less/structure/scrummaster.html>, 2017.
- [26] The LeSS Company. Teams. <https://less.works/less/structure/teams.html>, 2017.

- [27] 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.
- [28] Torgeir Dingsøy, Sridhar Nerur, VenuGopal Balijepally, and Nils Brede Moe. A decade of agile methodologies: Towards explaining agile software development. 2012.
- [29] Kathleen M Eisenhardt. Building theories from case study research. *Academy of management review*, 14(4):532–550, 1989.
- [30] Eric Evans. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2003.
- [31] Eric Evans. *Domain-Driven Design Reference: Definitions and Pattern Summaries*. Dog Ear Publishing, 2014.
- [32] Malte Foegen and Christian Kaczmarek. *Organisation in einer Digitalen Zeit: Ein Buch für die Gestaltung von reaktionsfähigen und schlanken Organisationen mit Hilfe von skalierten Agile Lean Mustern*. wibas GmbH, 2016.
- [33] Martin Fowler. Design - who needs an architect? *IEEE Software*, 20(5):11–13, Sept 2003.
- [34] Martin Fowler. Bounded context. <https://martinfowler.com/bliki/BoundedContext.html>, 2014.
- [35] Martin Fowler and Jim Highsmith. The agile manifesto. *Software Development*, 9(8):28–35, 2001.
- [36] Andreas Freitag, Florian Matthes, Christopher Schulz, and Aneta Nowobiliska. A method for business capability dependency analysis. In *International Conference on IT-enabled Innovation in Enterprise (ICITIE2011)*, Sofia, 2011.
- [37] Matthias Galster, Samuil Angelov, Marcel Meesters, and Philipp Diebold. A multiple case study on the architect’s role in scrum. In *Product-Focused Software Process Improvement: 17th International Conference, PROFES 2016, Trondheim, Norway, November 22-24, 2016, Proceedings 17*, pages 432–447. Springer, 2016.
- [38] Lynda Girvan and Debra Paul. *Agile and Business Analysis: Practical guidance for IT professionals*. BCS Learning and Development Limited, 2017.
- [39] Kenji Hiranabe. Modeling in the agile age: What to keep next to code to scale agile teams. <https://www.infoq.com/articles/kenji-modeling-agile>, 2013.
- [40] Scaled Agile Inc. *SAFe 4.0 Introduction: Overview of the Scaled Agile Framework for Lean Software and Systems Engineering (White Paper)*. scaledagile.com, 2016.

- [41] Scaled Agile Inc. Agile architecture. <http://www.scaledagileframework.com/agile-architecture/>, 2017.
- [42] Scaled Agile Inc. Domain modeling. <http://www.scaledagileframework.com/domain-modeling/>, 2017.
- [43] Scaled Agile Inc. Download the big picture. <http://www.scaledagileframework.com/posters/>, 2017.
- [44] Scaled Agile Inc. What's new in safe 4.5? <http://www.scaledagileframework.com/whats-new-in-safe-45//>, 2017.
- [45] iteratec GmbH. iteraplan - amazing eam. <https://www.iteraplan.de/en/iteraplan/>, 2017.
- [46] Ron Jeffries. *The Nature of Software Development*. Pragmatic Bookshelf, 2017.
- [47] Wolfgang Keller. Eam - quo vadis? *Objektspektrum - IT Management und Software-Engineering*, (4):46–51, 2017.
- [48] Simon Kneafsey. Scaled professional scrum and nexus - nexus+. <http://www.thescrummaster.co.uk/scrum/scaled-professional-scrum-and-nexus-nexus-plus/>, 2017.
- [49] Maarit Laanti. Characteristics and principles of scaled agile. In *International Conference on Agile Software Development*, pages 9–20. Springer, 2014.
- [50] Einar Landre, Harald Wesenberg, and Harald Rønneberg. Architectural improvement by use of strategic level domain-driven design. In *Companion to the 21st ACM SIG-PLAN symposium on Object-oriented programming systems, languages, and applications*, pages 809–814. ACM, 2006.
- [51] Craig Larman. *Scaling lean and agile development: thinking and organizational tools for large-scale Scrum*. Pearson Education India, 2008.
- [52] Craig Larman. Large-scale agile design and architecture: Ways of working. <https://www.infoq.com/articles/large-scale-agile-design-and-architecture>, 2011.
- [53] Craig Larman and Bas Vodde. Feature team primer. <http://www.featureteamprimer.org/#sthash.kVpNbrjx.ebVfdUfo.dpuf>, 2010.
- [54] Craig Larman and Bas Vodde. *Practices for scaling lean and Agile development: large, multisite, and offshore product development with large-scale scrum*. Pearson Education, 2010.

- [55] Craig Larman and Bas Vodde. *Large-Scale Scrum: More with LeSS*. Addison-Wesley Professional, 2016.
- [56] Dean Leffingwell. *Scaling software agility: best practices for large enterprises*. Pearson Education, 2007.
- [57] Dean Leffingwell. *Agile software requirements: lean requirements practices for teams, programs, and the enterprise*. Addison-Wesley Professional, 2011.
- [58] Dean Leffingwell. *Safe 4.0 reference guide: Scaled agile framework for lean software and systems engineering*. 2017.
- [59] Dean Leffingwell, Ryan Martens, and Mauricio Zamora. *Principles of agile architecture*. Leffingwell, LLC. and Rally Software Development Corp, 2008.
- [60] Ben Linders. *Nexus guide for scrum is published*. <https://www.infoq.com/news/2015/09/nexus-guide-scrum-published>, 2015.
- [61] Steven A. Lowe. *An introduction to event storming: The easy way to achieve domain-driven design*. <https://techbeacon.com/introduction-event-storming-easy-way-achieve-domain-driven-design>, 2017.
- [62] Garm Lucassen, Fabiano Dalpiaz, Jan Martijn E. M. van der Werf, and Sjaak Brinkkemper. *Improving agile requirements: the quality user story framework and tool*. *Requirements Engineering*, 21(3):383–403, 2016.
- [63] Christoph Mathis. *SAFe-Das Scaled Agile Framework: Lean und Agile in großen Unternehmen skalieren*. dpunkt. verlag, 2016.
- [64] Scott Millet. *Patterns, Principles and Practices of Domain-Driven Design*. John Wiley and Sons, 2015.
- [65] M. Paasivaara and C. Lassenius. *Scaling scrum in a large globally distributed organization: A case study*. In *2016 IEEE 11th International Conference on Global Software Engineering (ICGSE)*, pages 74–83, Aug 2016.
- [66] Inc. Pivotal Software. *Pivotal labs*. <https://pivotal.io/labs>, 2017.
- [67] Dobrila Rancic Moogk. *Minimum viable product and the importance of experimentation in technology startups*. <http://timreview.ca/article/535>, 2012.
- [68] Linda Rising and Norman S Janoff. *The scrum software development process for small teams*. *IEEE software*, 17(4):26–32, 2000.
- [69] Jeanne W. et. al. Ross. *Designing digital organizations*. 2016.

- [70] 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.
- [71] Agile Scaling. Ask: Agile scaling knowledge - the matrix. <http://www.agilescaling.org/ask-matrix.html>, 2014.
- [72] Ken Schwaber. Scrum development process. In *Business Object Design and Implementation*, pages 117–134. Springer, 1997.
- [73] Ken Schwaber. *Agile project management with Scrum*. Microsoft press, 2004.
- [74] Ken Schwaber. *The enterprise and Scrum*. Microsoft press, 2007.
- [75] Ken Schwaber. More on scaling scrum. <https://kenschwaber.wordpress.com/2015/01/30/more-on-scaling-scrum/>, 2015.
- [76] Scrum.org. Nexus guide - the definitive guide to nexus: The exoskeleton of scaled scrum development. <https://www.scrum.org/resources/nexus-guide>, 2015.
- [77] Gernot Dr. Starke. arc42. <http://arc42.org/>, 2017.
- [78] TechBeacon. Enter the nexus: Ken schwaber on scaling scrum and the future of agile. <https://techbeacon.com/enter-nexus-ken-schwaber-scaling-scrum-future-agile>, 2017.
- [79] CA Technologies. Ca technologies: Most businesses will be software-driven in 36 months. <https://www.ca.com/us/company/newsroom/press-releases/2015/ca-technologies-most-businesses-will-be-software-driven-in-36-months.html>, 2015.
- [80] Ömer Uludağ, Martin Kleehaus, Xian Xu, and Florian Matthes. Investigating the role of architects in scaling agile frameworks. In *21th Conference on Enterprise Distributed Object Computing (EDOC), Québec City, Canada*, 2017.
- [81] Edward Uy and René Rosendahl. Migrating from sharepoint to a better scrum tool. In *Agile, 2008. AGILE'08. Conference*, pages 506–512. IEEE, 2008.
- [82] Ravi Verma. I'm not calling your baby ugly - two ways and 25 dimensions to compare agile scaling frameworks. <http://smoothapps.com/index.php/2017/04/im-not-calling-your-baby-ugly-2-ways-and-25-dimensions-to-compare-agile-scaling-frameworks-less-safe-nexus/>, 2017.
- [83] Vaughn Vernon. *Implementing domain-driven design*. Addison-Wesley, 2013.
- [84] Vaughn Vernon. *Domain-driven design distilled*. Addison-Wesley Professional, 2016.

- [85] Jan Vom Brocke, Alexander Simons, Bjoern Niehaves, Kai Riemer, Ralf Plattfaut, Anne Cleven, et al. Reconstructing the giant: On the importance of rigour in documenting the literature search process. In *ECIS*, volume 9, pages 2206–2217, 2009.
- [86] Gerard Wagenaar, Sietse Overbeek, and Remko Helms. Describing criteria for selecting a scrum tool using the technology acceptance model. In *Asian Conference on Intelligent Information and Database Systems*, pages 811–821. Springer, 2017.
- [87] Peter Weill and Stephanie L Woerner. Thriving in an increasingly digital ecosystem. *MIT Sloan Management Review*, 56(4):27, 2015.
- [88] Harald Wesenberg, Einar Landre, and Harald Rønneberg. Using domain-driven design to evaluate commercial off-the-shelf software. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 824–829. ACM, 2006.
- [89] Eberhard Wolff. *Microservices: Flexible Software Architecture*. Addison-Wesley Professional, 2016.
- [90] Robert K Yin. *Case study research: Design and methods*. Sage publications, 2013.