Empirical Studies to Identify Best Practices for Addressing Recurring Concerns of Enterprise Architects and Solution Architects in Large-Scale Agile Development

NIKLAS REITER, Technische Universität München, Germany

Over the past decades, the emergence of agile development methods has transformed the way software is developed. Even though systems are getting more and more complex, companies have to develop and release software faster and at the same time increase the quality. Due to the proven success of agile methods, companies also try to make use of these benefits in large-scale agile software development projects. However, this represents a risk and is often associated with challenges such as managing silos, complex functional dependencies between systems and establishing an agile way of working for multiple teams. Especially enterprise and solution architects face a large number of problems. Regardless of their importance, there is a lack of research on the concerns and best practices involved. Based on a mixed-methods research design we provide an overview of typical concerns and present five best practices of enterprise and solution architect in large-scale agile development.

CCS Concepts: • **Software and its engineering → Agile software development**.

Additional Key Words and Phrases: concerns, large-scale agile development, patterns

ACM Reference Format:
Niklas Reiter. 2019. Empirical Studies to Identify Best Practices for Addressing Recurring Concerns of Enterprise Architects and Solution Architects in Large-Scale Agile Development. 1, 1 (September 2019), 24 pages. https://doi.org/0000001.0000001_2

1 Introduction

The increasingly rapid and frequent changes in market conditions, technical and regulatory changes and the necessity to reduce costs, force large IT organizations to carry out complex business transformations at more and more frequent intervals [Ross et al. 2006]. Software development is playing an increasingly important role, as companies are forced to develop software faster and react to changing customer requirements [Besson and Rowe 2012; Gerster et al. 2018]. For years, methodologies such as Waterfall or the Spiral Model have been used to develop software and create business value [Boehm 1988; Royce 1987]. However, these methodologies often involve long planning phases with a Big Design Up Front (BDUF) and large documentation efforts and are not designed to adapt to changing requirements, unexpected events or even the interaction with customers within a software development project [Schwaber 1997]. Therefore, it is not without reason that the use of agile methods has strongly increased over the last two decades [Maiden and Jones 2010; VersionOne 2019]. Especially agile methods such as Extreme Programming [Beck and Gamma 2000] and Scrum [Schwaber and Beedle 2002] are able to solve these challenges by emphasizing collaboration within teams, active customer involvement, change tolerance and iterative delivery of features [Dingsøyr and Moe 2014a; Kettunen 2007]. They are designed for small, self-organized and collaborative teams that work closely with customers, incrementally develop software products and deliver multiple releases within a project [Boehm and Turner 2005a; Dyba and Dingsoyr 2009].

Nevertheless, the usage of agile methods in the context of large-scale agile development is difficult as they introduce new challenges such as the coordination between teams [Petersen and Wohlin 2010], the right balance between emergent and intentional architecture [Dingsøyr et al. 2014; Uludağ et al. 2017], handling distributed projects and lack of know-how on how to do architecting for managing and building complex software systems [Dingsøyr and Moe 2014b; Leffingwell 2007; Rost et al. 2015; Uludağ et al. 2018]. In addition, the role of architects in agile methods is often not described explicitly which is not surprising as the Agile Manifesto states that the best architecture emerges from self-organizing teams, mainly through coding and refactoring activities [Beck et al. 2001].

Therefore, the role of enterprise and solution architects in this context is very important as they have to maintain a high-level and holistic vision of enterprise solutions and development initiatives, align individual program and product strategies with enterprise objectives, report technology and architecture requirements / issues to enterprise architects for alignment and issue resolution and work closely with ATs by providing guidance through business and technical road maps are becoming increasingly important [Uludag et al. 2019b; Uludağ et al. 2017]. Despite the relevance of enterprise and solution architects for large-scale agile software development, current concerns and best practices are not covered by literature yet. This paper aims to fill this research gap by providing a collection of recurring concerns and best practices of enterprise and solution architects based on a mixed-methods research approach.

The remainder of this paper is structured as follows. Section 2 describes the research approach following the pattern-based design research (PDR) method. In Section 3 an overview of related work in the field of large-scale agile software development is given. In addition, related patterns languages are described. It follows an elaboration of the large-scale agile development pattern language in Section 4. In Section 5 we provide an overview of identified concerns and best practices and present five patterns. Section 6 contains a discussion of our main findings. Finally, in Section 7, a summary of the results and an outlook for possible future research in this area is given.

Author's address: Niklas Reiter, Technische Universität München, Garching bei München, 85748, Germany, niklas.reiter@tum.de, matthes@tum.de.

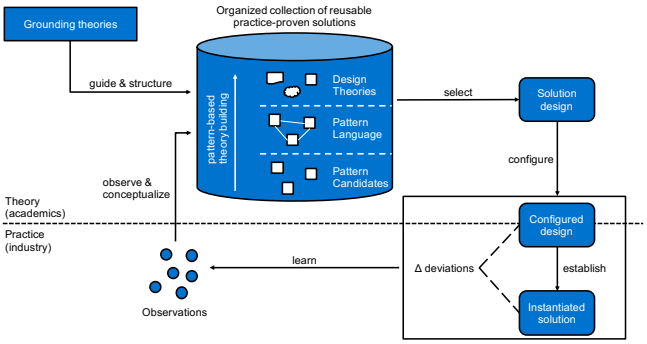

Fig. 1. Pattern-based design research [Buckl et al. 2013]

2 Research Approach

The goal of this paper is to document best practices that address recurring concerns of enterprise and solution architects in large-scale agile development. For this purpose, we followed the pattern-based design research (PDR) method as recommended by [Buckl et al. 2013] to balance the rigor and relevance of the research. The PDR method builds on established concepts such as design theory and patterns and allows researchers to theorize and learn from the intervention at the industry partners while conducting developmental research. As shown in Fig. 1, the PDR method consists of four phases: *observe & conceptualize*, *pattern-based theory formation & nexus instantiation*, *solution design & application* and *evaluation & learning*. Within the *observe & conceptualize* phase, best practices for recurring concerns are observed and documented according to a typical pattern structure (see Section 4). In phase *pattern-based theory formation & nexus instantiation* these pattern candidates are then conceptualized using grounding theories. A pattern can develop from this if the *Rule of Three* [1] applies. This states that a documented pattern must refer to at least three known applications in practice to ensure the reusability of the solution provided [Coplien 1996], which are then integrated into the large-scale agile development pattern language. Design theories can be developed by documenting appropriate context and problem descriptions. Pattern candidates, patterns, the pattern language and design theories together form an organized knowledge base of reusable and proven solutions. In the *solution design & application* phase, typical stakeholders select patterns based on their individual concerns and adapt them to the specific terminology of the company. After their configurations, patterns can be defined within the case society. During the *evaluation & learning* phase, deviations between the actual and the original pattern configuration are recognized and documented. With the help of the deviations, patterns or best practices can be either identified or developed further.

3 Related Work and Pattern Languages

The following section provides an overview of current pattern languages for agile and / or large-scale agile software development. An overview created by the authors of the LSADPL ([Uludag et al. 2019a]) was taken as a foundation and extended by missing sample languages based on an extensive literature research.

The number of current scientific papers on this topic is low, although the question "Architecture and Agility - How much Design is Sufficient for Different Problem Classes" was already classified by researchers as the second most burning question at the XP Conference in 2010 [Freudenberg and Sharp 2010]. Since then, the topic of agility has gained further importance in large-scale agile development projects as a study by Version One has confirmed. According to the 12th survey, more than 51% of all interviewees work with agile methods [VersionOne 2018]. This emphasizes the relevance of this research area. Nevertheless, there is still a lack of knowledge, especially about challenges and success factors [Alsaqaf et al. 2019; Uludag et al. 2018].

In 2016, researchers began to investigate large-scale agile transformations. Among other things, 35 challenges and 29 success factors from a sample of 42 organizations were described in a systematic overview [Dikert et al. 2016]. While *requirement engineering* and *change resistance* were identified as the most critical challenges, soft factors such as *coaching and training of agile models / methods* and *management support* played a key role among the greatest success factors. These results were confirmed in a literature search conducted by [Kalenda et al. 2018]. For a validation in practice, a software company was specifically selected that currently worked on the scaling of agile methods. Thereby the following challenges were identified: *Complete organization-wide integration*, *quality assurance*, *change resistance* and *speed of implementation strategy*. Further success factors are: *Corporate culture*, *hands-on experience* and *agile views and values*. [Uludag et al. 2018] conducted a comprehensive analysis of a total of 73 papers. The researchers identified around 14 different typical stakeholders and 79 challenges. The latter were grouped into 11 categories. Tab. 2 describes an overview of related patterns languages.

4 Large-Scale Agile Development Pattern Language

With the usage of agile methods in large-scale agile development projects, new challenges arise. First, due to the involvement of an increased number of entities/stakeholders, communication and coordination is more complex and requires greater effort. Second, the more products are involved in a project, the higher is the number of dependencies which makes the architectural implementation/integration difficult [Boehm and Turner 2005b; Dikert et al. 2016; Paasivaara and Lassenius 2014]. In order to solve these problems and to fully exploit the advantages of agility in large environments [Kettunen and Laanti 2008] there are first attempts to solve these challenges with scalable agile frameworks [Alqudah and Razali 2016; Dingsøyr et al. 2019]. Other methods, such as research studies that explain how to meet challenges of large-scale agile development, are difficult to find in the literature today. In order to address this gap, the goal is to identify recurring concerns and document best practices in this context based on the idea of [Alexander 1977]. This is followed by a description of the structure of the pattern language (see Fig. 2).

The latter distinguishes three different types of patterns:

[1] The *rule of three* states that a documented pattern must refer to at least three known uses in practice to ensure the re-usability of the provided solution.

Table 1. Overview of Related Pattern Languages

Source	Scope & Goal	Focus on agile development	Number of patterns	Pattern categories	Pattern example
[Coplien 1995]	Collection of patterns for shaping a new organization and its development processes	Partially	42	(1) Process Patterns; (2) Organizational Patterns;	- CODE OWNERSHIP - GATEKEEPER - FIRE WALLS
[Harrison 1996]	Collection of patterns for creating effective software development teams	No	4	-	- UNITY OF PURPOSE - DIVERSITY OF MEMBERSHIP - LOCK'EM UP TOGETHER
[Ambler 1998]	Patterns for medium to large-scale object-oriented software development	Partially	18	(1) Task Process Patterns; (2) Stage Process Patterns; (3) Phase Process Patterns;	- ENHANCE AND ENSURE QUALITY OF TECHNICAL REVIEW - ITERATIVE PROGRAMMING TASKS / ACTIVITIES - ONLY BUILD WORKING AND TESTABLE SOFTWARE
[Beedle et al. 1999]	Collection of Scrum Patterns	Yes	3	-	- SPRINT - BACKLOG - SCRUM MEETINGS
[Taylor 2000]	Collection of patterns for creating product software development environments	No	9	(1) Establishing a Production Potential; (2) Maintaining a Production Potential; (3) Preserving a Production Potential;	- DELIVERABLES TO GO - PULSE - BOOTSTRAPPING
[Coplien and Harrison 2004]	Collection of organizational patterns that are combined into a collection of four pattern languages	Yes	94	(1) Project Management; (2) Piecemeal Growth; (3) Organizational Style; (4) People and Code;	- SKILL MIX - DEMO PREP - FEW ROLES
[Bozheva and Gallo 2005]	Patterns originating from different agile methodologies with special focus on rationale for applying the patterns	Yes	39	(1) Implementation & Testing; (2) Design; (3) Resource Organization; (4) Contract Management; (5) Software Process Improvement; (6) Project and Requirements Management;	- INGRIDIENT GATHERING - CUSTOMER COMMUNICATION - REVIEWING WORK PRODUCTS - BUILD MODELS (PROTOTYPES)
[Elssamadisy 2007]	Collection of patterns of Agile Practice Adoptions	Yes	9	(1) Pattern to Business Value Mappings; (2) Pattern to Smell Mappings;	- AUTOMATED DEVELOPER TESTS - TEST-LAST AND TEST-FIRST DEVELOPMENT - CONTINOUS INTEGRATION - SIMPLE DESIGN - AUTOMATED FUNCTIONAL TESTS - COLLECTIVE CODE OWNERSHIP
[Elssamadisy 2008]	Collection of patterns for successfully adopting agile practices	Yes	38	(1) Feedback Practices; (2) TechnicalPractices; (3) Supporting Practices; (4) The Clusters;	- REFACTORING - CONTINUOUS INTEGRATIO - SIMPLE DESIGN
[Beedle et al. 2010]	Collection of the most essential best practices of Scrum	Yes	11	-	- DAILY SCRUM - SPRINT BACKLOG - SPRINT REVIEW
[Keutzer et al. 2010]	Pattern Language for Engineering (Parallel) Software, that can help to increase quality and performance by pulling the essential set of design patterns	Yes	56	(1) Structural Patterns; (2) Computational Patterns; (3) Algorithm strategy Patterns; (4) Implementation Strategy Patterns; (5) Parallel execution Patterns;	- ITERATIVE REFINEMENT - EVENT BASED, IMPLICIT INVOCATION - BACKTRACK BRANCH AND BOUND - RECURSIVE SPLITTING - STRICT DATA PAR - TASK GRAPH - COLLECTIVE COMMUNICATION
[Lescher 2010]	Patterns to build a globally distributed team and make communication and collaboration effective	Partially	5	(1) Kick-Off and Teambuilding; (2) Project Communication;	- FOCUS ON DIRECT COMMUNICATION - ESTABLISH COORDINATION MECHANISM - CREATE COMMUNITIES
[Välimäki 2011]	Enhancing performance of project management work through improved global software project management practice	Partially	18	(1) Directing a Project; (2) Starting up a Project; (3) Initiating a Project; (4) Controlling a Stage; (5) ManagingStage Boundaries; (6) Closing a Project; (7) Managing Product Delivery; (8) Planning;	- COLLOCATED KICK-OFF - CHOOSE ROLES IN SITES - ITERATION PLANNING
[Monasor et al. 2013]	Pattern language for pedagogical patterns in Global Software Development, that can help to build a custom set of practices	No	-	(1) Cultural Patterns; (2) Communication and Interaction Patterns; (3) Patterns for Project Management; (4) Testing Patterns; (5) Requirements Engineering Patterns	- UNPRODUCTIVE PRODUCTIVITY - HESITANT TO ALWAYS SAY YES - OWNING RATHER THAN MODULARIZING - MULTI -LEVEL DAILY MEETINGS - SYNCHRONIZED TEST ENVIRONMENTS - USE BIDIRECTIONAL CROSS REFERENCES
[Sutherland et al. 2014]	Pattern language for Hyperproductive Software Development. Patterns for problems of Scrum teams that hinder them to finish early	Yes	No	(1) Getting Ready for a Sprint; (2) Dealing with Disruptive Problems During a Sprint; (3) Becoming Hyper-Productive;	- STABLE TEAMS - YESTERDAY'S WEATHER - SWARMING: ONE PIECE CONTINUOUS FLOW - HAPPINESS METRIC - TEAMS THAT FINISH EARLY ACCELERATE FASTER
[Kausar and Al-Yasiri 2015]	Patterns for distributed teams using agile methods with special focus on offshoring scenarios	Yes	15	(1) Management Patterns; (2) Communication Patterns; (3) Collaboration Patterns; (4) Verification Patterns;	- SCRUM OF SCRUM - FOLLOW THE SUN - COLLABORATIVE PLANNING POKER - CENTRAL CODE REPOSITORY - LOCAL PAIR PROGRAMMING
[Ruy et al. 2015]	Pattern language to provide a stronger sense of connection between the patterns due to expressing relations such as dependence, temporal precedence of application	Partially	27	(1) Patterns of Work Units; (2) Patterns of Human Resources; (3) Patterns of Work Products;	- STAKEHOLDER ALLOCATION - CLEAR STAKEHOLDER DEFINITION - ORGANIZATIONAL / PROJECT TEAM DEFINITION - CLEAR ROLE DEFINITION OF TEAMS
[Javed et al. 2016]	Pattern language for construction and maintenance of software architecture traceability links to requirements and source code	Partially	5	-	- INITIAL TRACEABILITY CONSTRUCTION - TRACEABILITY COMPLETION - CONTINOUS TRACEABILITY MAINTENANCE - ON-DEMAND TRACEABILITY MAINTENANCE - TRACEABILITY QUALITY CHECKS
[Mitchell 2016]	Collection of patterns to address agile transformation problems	Yes	54	(1) Patterns of Method; (2) Patterns of Responsibility; (3) Patterns of Representation; (4) Anti-Patterns;	- LIMITED WIP - KANBAN SANDWICH - CONTROLLED FAILURE
[ScrumPLoP 2019]	Body of pattern literature around agile and Scrum communities	Yes	234(10)	(1) Value Stream; (2) Team; (3) Sprint; (4) Process Improvement; (5) ProductOrganization; (6) Distributed Scrum; (7) Scaling Scrum; (8) Scrum Core; (9) Misc;	- SCRUM MASTER - SCRUM OF SCRUMS - PORTFOLIO STANDUP
[Uludag et al. 2019a]	Collection of recurring concerns and patterns of typical stakeholders in large-scale agile development	Yes	70	(1) Culture and Mindset; (2) Enterprise Architecture; (3) Geographical Distribution;	- STRICTLY SEPARATE BUILD AND RUN STAGES - COMMUNITY OF PRACTICE - ITERATION DEPENDENCY MATRIX - DON'T USE AGILE AS A GOLDEN HAMMER

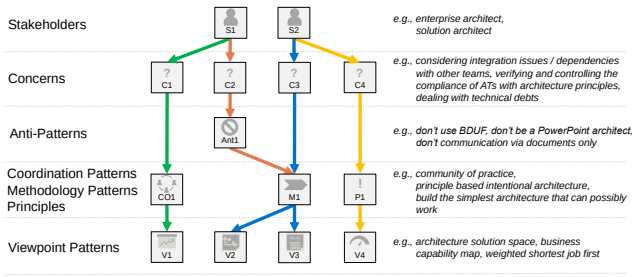

Fig. 2. Conceptual overview of the proposed pattern language

- **Coordination Patterns (CO-Patterns)** define coordination mechanisms that are proven solutions for recurring coordination problems such as dependencies between activities or the management of tasks or resources.
- **Methodology Patterns (M-Patterns)** define concrete steps that are proven solutions to a problem.
- **Viewpoint Patterns (V-Patterns)** define proven solutions for visualizing information such as documents, boards, metrics, models, and reports.

In addition, the pattern language contains four additional concepts:

- **Anti-Patterns** define solutions that are unfavourable or harmful to the success of a software project. Anti-patterns represent the counterpart to patterns.
- **Principles** provide a common direction for action with the help of rules and guidelines to address specific concerns.
- **Stakeholders** are defined as persons who have an interest in the project and/or are actively involved in the large-scale agile development [Uludağ et al. 2018].
- **Concerns** describe challenges of stakeholders. They can be categorized as different topics such as risks or responsibilities [42010:2011(E) 2011] and addressed by different **Patterns**, **Anti-Patterns** or **Principles**.

In the literature different pattern forms exist. Among the most common pattern forms are the Alexandrian Form, Gang of Four Form, Coplien Form and the Fowler Form [Ernst 2010; Fowler 2006]. Each pattern form has certain advantages and disadvantages [Ernst 2010] depending on the context and field of application. This means that there is no optimal form. When selecting or creating a new pattern form, a researcher should pay special attention to the context and the goal and incorporate his own experiences [Buschmann et al. 2007b; Ernst 2010]. According to [Fowler 2006] the selection process is a personal decision and should also consider one's writing style and the ideas to be conveyed. The template defined by [Buschmann et al. 1996; Ernst 2010] provides a good basis for large-scale agile development patterns. Fig. 3 describes the underlying conceptual model. It contains the key elements with which concepts and patterns of the model language are documented. Each element has a *identifier* and a *name* which simplifies referencing. There is also a section for stakeholders with a list of synonyms and related names called *alias*. For organizations there are two more sections called *category* and *scaling level*. It describes the category on the one hand and the organizational level at which a group company operates on

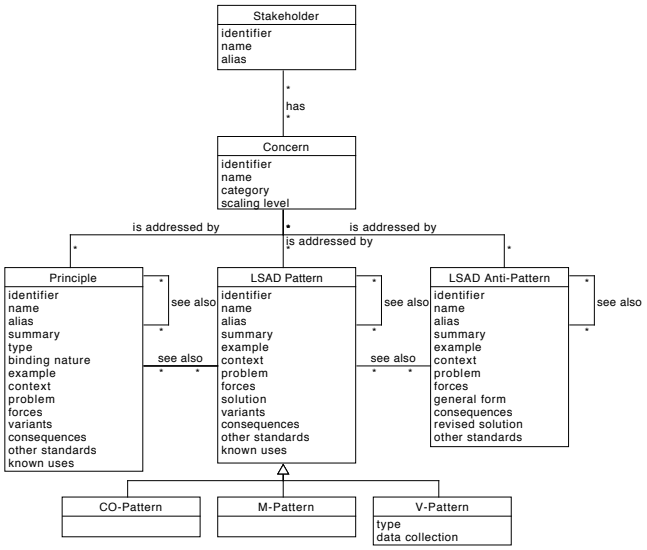

Fig. 3. Conceptual model of the Large-Scale Agile Development Pattern Language [Uludag et al. 2019a]

the other. In addition, principles, patterns and anti-patterns consist of eight common sections: First, in section *example*, a current practical example in the form of a project or product in which this pattern is/was used is illustrated. This is followed by a description of the conditions under which the problem occurs in section *context*. In section *problem* the problem itself is described in more detail. In section *forces* the question why the problem is difficult to solve is answered. Section *summary* describes how the problem should be solved. The *consequences* section lists the advantages and disadvantages associated with the solution (=pattern), while the optional *other standards* and *see* sections contain references to other solutions and frameworks. The *alias* section contains a list of synonyms. Principles and patterns consist of both *variants* and *known uses* which list variants and alternatives as well as proven applications in practice. The *type* and *binding nature* sections explicitly indicate whether the topics are recommended or mandatory by principles. The *solution* section explains the recommended solution for a pattern. In addition, there are specific sections, such as *general form* and *revised solution*, which apply only to anti-patterns. These include approaches that reinforce the problem at hand. For the V-patterns described above, the *type* and *data collection* sections provide a way to list the visualization concept and the collection processes required to create it. To illustrate the maturity of a pattern, identified patterns are marked with a star notation similar to [Buschmann et al. 2007a]. Depending on the number of stars, the effect of a pattern for a problem is lower or higher. The higher the number of stars, the more effective a pattern is for a particular problem. More precisely, the marking with no star symbolizes that the pattern can be a solution for an observed problem but must still be significantly revised in order to achieve an effectiveness. A star means that the pattern addresses a real problem but has yet to mature. Two stars indicate that the pattern in its current form effectively addresses a problem.

Table 2. Overview of Related Pattern Languages

Step	Research Design	Approach according to	Goal	Outcome related to this paper	Published Articles
1	Structured Literature Review	[Vom Brocke et al. 2009]	Identifying recurring concerns of stakeholders in large-scale agile development	List of 26 recurring concerns of enterprise and solution architects	[Uludağ et al. 2018]
2	Case Study	[Runeson and Höst 2009; Yin 2017]	Investigating the role of enterprise architects in large-scale agile development and their collaboration with agile teams	List of X recurring concerns and X good practices and pattern candidates of enterprise architects	[Uludag et al. 2019a], [Uludag et al. 2019b]
3	Expert Interviews	[Bogner et al. 2009]	Identifying recurring concerns and best practices of enterprise and solution architects in large-scale agile development	List of X recurring concerns and X good practices and pattern candidates of enterprise and solution architects	-

Table 3. Overview of interview partners

Organization	No. Case Study interviews	No. Expert interviews	Roles
CarCo	20	3	Chief Technology Officer, Enterprise Architect, Group Lead IT, Product Owner, Requirements Engineer, Solution Architect, Scrum Master
RetailCo	5	3	Chapter Lead Business Process Architecture, Chief Scrum Master, Enterprise Architect, Product Owner, Solution Architect, Scrum Master,
GlobalInsureCo	12	-	Agile Developer, Chapter Lead Agile Coaching, Enterprise Architect
TechCo	-	3	Enterprise Architect, Solution Architect
ConsultCo	-	1	Solution Architect
SoftCo	-	3	Enterprise Architect, Solution Architect
ITCo	4	-	Enterprise Architect, Product Owner
PublicInsureCo	4	-	Agile Developer, Enterprise Architect, Head of IT Governance, Head of IT Governance Department

5 Recurring Concerns and Best Practices

In order to identify recurring concerns and best practices, a mixed-methods research design in line with the recommendations of [Tashakkori and Teddlie 2010] was applied consisting of three distinct phases (see Tab. 2).

At first a structured literature review according to [Vom Brocke et al. 2009] was conducted to identify recurring concerns of stakeholders in large-scale agile development [Uludağ et al. 2018]. The latter were summarized in a list which formed the basis for the next steps. In step two, the [Runeson and Höst 2009; Yin 2017] case study approach was used. With the help of semi-structured interviews the relevance of the elements identified from the literature was examined and validated in practice. New or previously unknown challenges and best practices were included in the list and used in subsequent interviews. The approach to conduct semi-structured interviews was specifically chosen to allow interviewees to answer questions as openly as possible and without any guidance. This was achieved through a combination of open and closed questions. In step three, expert interviews according to [Bogner et al. 2009] were performed. Based on the gathered data from previous steps, recurring concerns, best practices and pattern candidates of enterprise and solution architects were identified. In order to facilitate the triangulation of the observers, an interview was usually conducted by at least two researchers in a personal conversation [Runeson and Höst 2009]. Tab. 3 gives a detailed overview of the number of interviews.

As shown in Fig. 4, a total of 35 recurring concerns were observed by developers, enterprise architects and solution architects. 16 out of 35 concerns have already been identified in the structured literature review and confirmed as a concern by the case study and expert interviews. Within the case study, additional 9 concerns were identified and added. By conducting expert interviews, another 10 previously unknown concerns could be identified and added to the list.

Fig. 5 provides an overview of identified best practices. In total, we identified 65 pattern candidates consisting of 16 Principles, 3 Coordination-Patterns, 19 Methodology-Patterns, 18 Viewpoint-Patterns and 9 Anti-Patterns. The application of the rule of three according to [Coplien 1996], resulted in a total of 43 patterns consisting of 11 Principles, 3 Coordination-Patterns, 12 Methodology-Patterns, 10 Viewpoint-Patterns, and 7 Anti-Patterns.

Fig. 6 shows the current version of our large-scale agile development language which is part of a specially defined prototypical web application and can be found under [2]. Furthermore five nodes are highlighted which represent the best practices that are described in section 5. More information about the large-scale agile development patterns and concepts can be found in Appendix A.

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

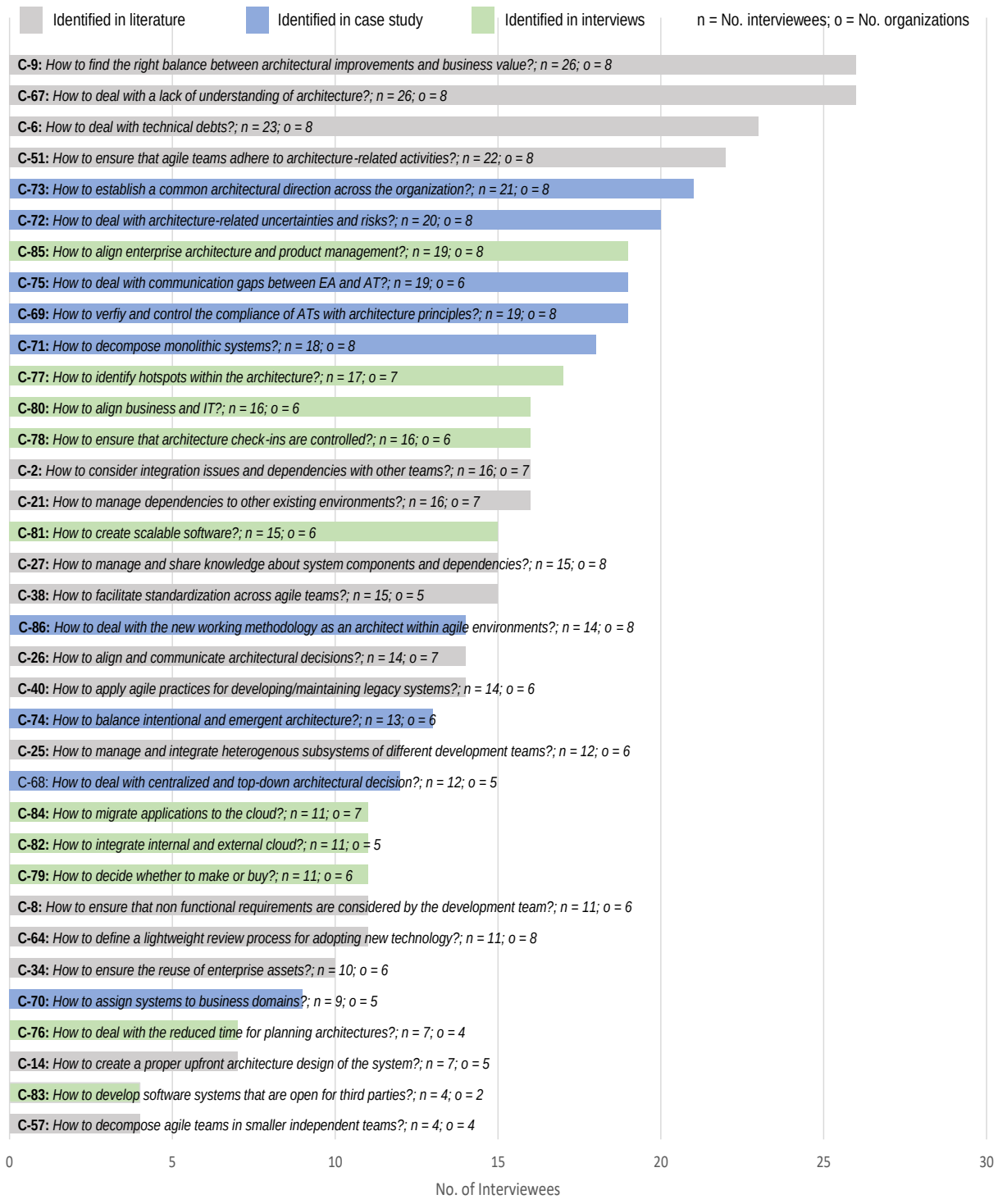

Fig. 4. Recurring concerns identified

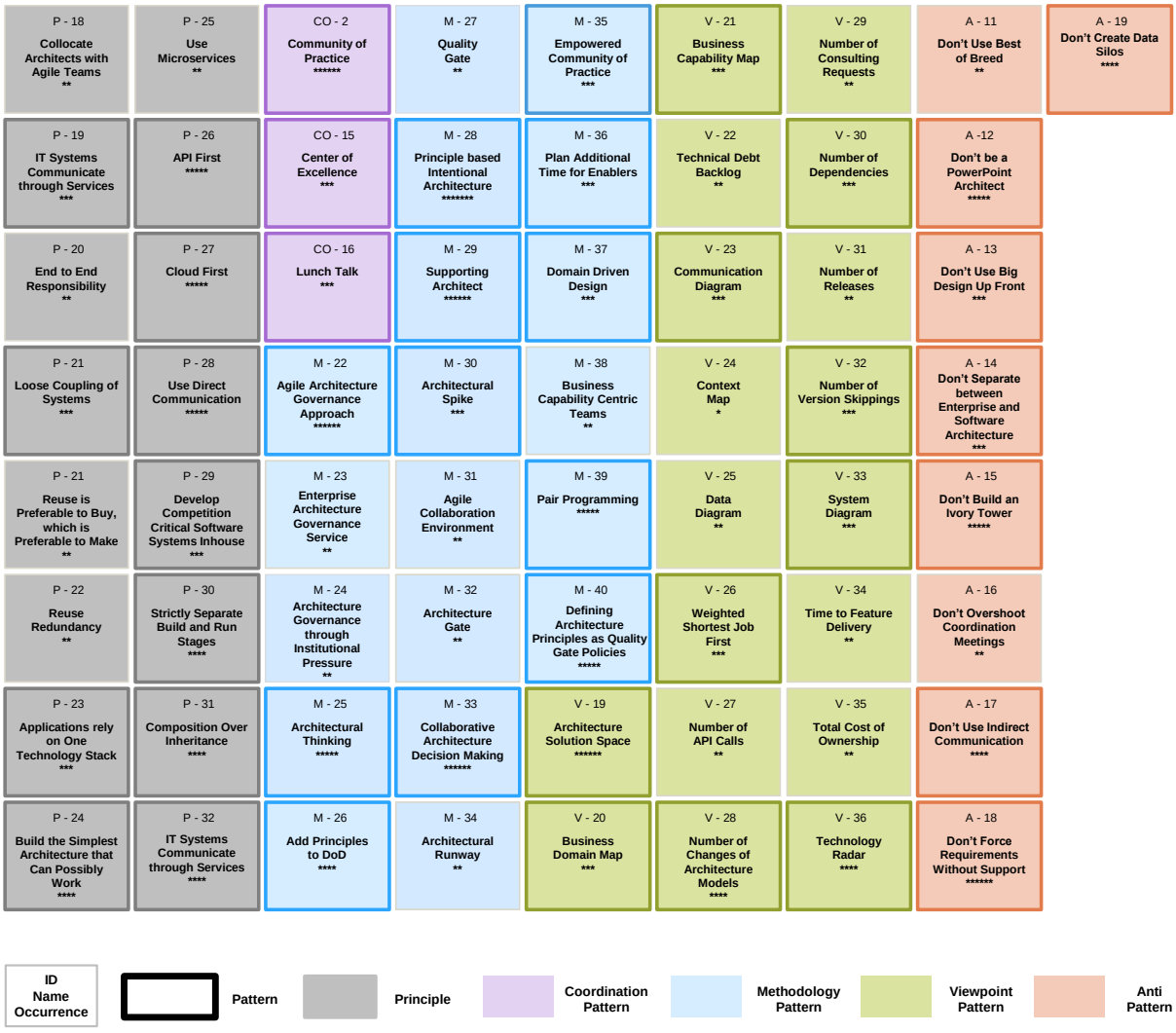

Fig. 5. Recurring Principles, Patterns and Anti-Patterns identified

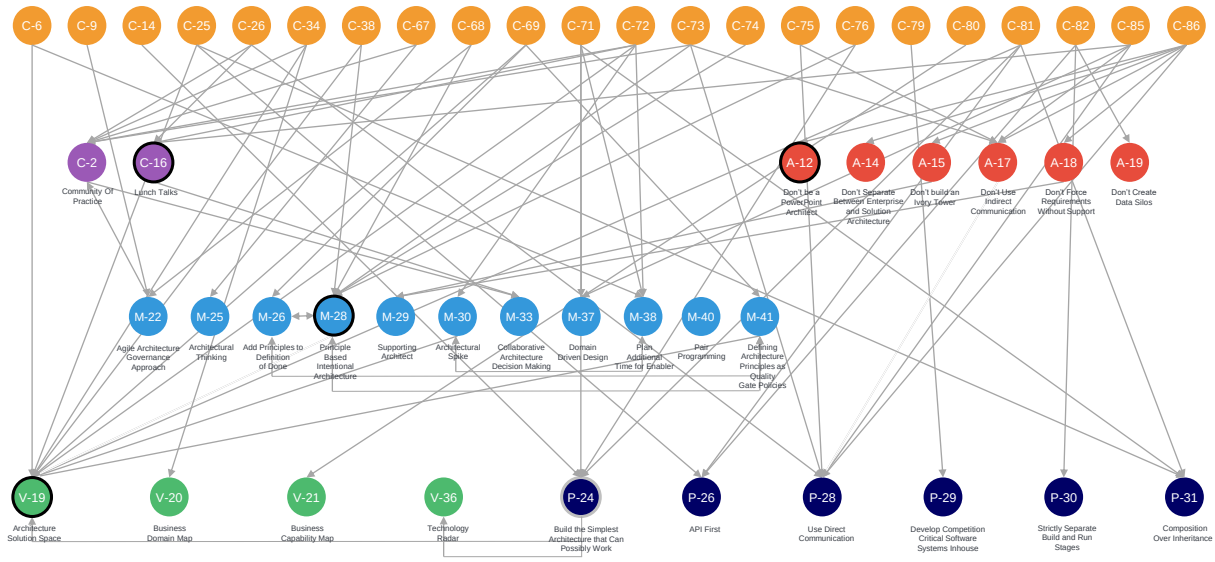

Fig. 6. Pattern language for enterprise and solution architects *

5.1 Principle: Build the Simplest Architecture that can Possibly Work (P-24) *

Principle Overview	
Alias	Simplest Architecture
Summary	Build the Simplest Architecture that can Possibly Work ensures that the creation of an architecture does not consume too much time and causes no delay of development.
Type	Software Architecture
Recommended	Binding Nature

5.1.1 Example

Five agile teams of RetailCo develop a new product. At the beginning of the project, an architect is asked to create and provide a target picture as quickly as possible. Due to the trade-off between quality vs. time, the architect struggles.

5.1.2 Context

Constantly changing requirements in large-scale agile development force developers and architects to adapt or even discard software architectures at more and more frequent intervals. Even though agile teams commit to agile software development, it happens often, that architects try to build the optimal architecture or lose themselves in details. This is problematic, because detailed plannings delay the start of development and result in a decrease of velocity. Therefore, architects need to be able to create temporary architectures in a short amount of time.

5.1.3 Problem

The following concerns are addressed by Build the Simplest Architecture that can Possibly Work:

- *C-14: How to create a proper upfront architecture design of the system?*
- *C-71: How to decompose monolithic systems?*
- *C-76: How to deal with the reduced time for planning architectures?*
- *C-81: How to create scalable software?*

5.1.4 Forces

The following forces influence Build the Simplest Architecture that can Possibly Work:

- *Attention to detail*: Architects tend to spend a lot of time working on one single architecture to make it perfect which delays the start of development.
- *Tradeoff between quality and time*: When creating architectures in agile environments, there is always the tradeoff between quality and time because time for planning architectures is limited. This especially affects cross-cutting aspects like security and scalability of a product because it involves all architecture layers and is either achieved in total or not.
- *Lack of planning and conception phase*: Due to release pressure and constant change, time is critical which makes it difficult for architects to cover all aspects, uncertainties and risks in detail and at the same time keep the development pace.

5.1.5 Variants

A variant of Build the Simplest Architecture that can Possibly Work could not be identified.

5.1.6 Consequences

The following benefits of Build the Simplest Architecture that can Possibly Work are known:

- Development of software is faster
- More time for support and enablement of agile teams

The following liabilities of Build the Simplest Architecture that can Possibly Work are known:

- Quality of architecture decreases
- Architecture-related uncertainties and risks increase

5.1.7 See Also

In order to create a software architecture according to Build the Simplest Architecture that can Possibly Work, the following V-Patterns should be considered:

- V-19: Architecture Solution Space
- V-36: Technology Radar

5.1.8 Other Standards

An alternative would be to build the software architecture based on agile principles. Furthermore, architects should use iterative development processes as applied in *The Agile Model Driven Development (AMDD) lifecycle for software projects* to develop architectures in agile environments [Larman and Basili 2003; Ruparelia 2010].

5.1.9 Known Uses

The following uses of Build the Simplest Architecture that can Possibly Work are known:

- CarCo
- ItCo
- PublicInsureCo
- RetailCo

5.2 CO-Pattern: Lunch Talks (CO-16) *

CO-Pattern Overview	
Alias	Business Lunch
Summary	A Lunch Talk provides an option to schedule ad hoc meetings at a time where participants are per se highly available (i.e. lunch time).

5.2.1 Example

SoftCo is in the middle of a tight scheduled software development project. After a meeting with the customer, a major technical requirement needs to be changed. The responsible solution architect does not know how to implement the change properly into the existing solution. In order to get help, he tries to organize a meeting but due to tight schedules, most of the experts are not available during regular business hours.

5.2.2 Context

Developers and architects are bound to a tight schedule, especially in an agile environment. In critical times, for example when an upcoming release is close or if a system is causing problems every day's agendas are fully booked. 100% of the capacity is allocated and there remains no time for additional and ad hoc meetings.

5.2.3 Problem

The following concerns are addressed by Lunch Talks:

- *C-25: How to manage and integrate heterogeneous subsystems of different development teams?*
- *C-26: How to align and communicate architectural decisions?*
- *C-72: How to deal with architecture-related uncertainties and risks?*
- *C-85: How to align enterprise architecture and product management?*

5.2.4 Forces

The following forces influence Lunch Talks:

- *Co-location is not always guarantied*: Often, members of the same development team are not located at the same physical location which makes face-to-face communication and collaboration among the team members difficult.
- *Difficult to schedule ad hoc meetings in an agile environments*: Architects and developers are bound to tight schedules, which makes it difficult to schedule meetings. But, important architectural challenges or problems (e.g. security) need to be taken care of quickly. Often, there is no time left to schedule a meeting in advance for the next couple of weeks. Time is critical and during lunch break, no one is occupied or in a meeting. Due to the fact, that a break has a defined time box, ad hoc lunch talks are perfectly suited for such matters. In addition, it does not slow down or restrict the work of the participants involved in any way.

5.2.5 Solution

In order to have a successful/Setting up a Lunch Talk some rules should be considered.

- *Have a clear objective:* The discussion should mainly focus on working practices and stick to the defined objective. In the best case, different best practices are discussed leading to a working solution.
- *Only invite experts:* Only invite colleagues that are experts in that specific field of discussion and know what they talk about.
- *Keep the number of people as small as possible*: Keeping the number of participants as small as possible reduces the probability to ramble off and talk about theoretical concepts.
- *Keep the meeting short:* In order to achieve the defined objective, a meeting should be short. Participants should try to achieve the goal in the given time. Furthermore, keeping the meeting short increases the motivation to participate.
- *Choose regular lunch time and place:* When setting up the meeting, regular lunch times should be considered. Furthermore, the place should be in close distance to the office to keep travel times to a minimum. A suitable place for a *Lunch Talk* is the organizations canteen.

5.2.6 Variants

A Lunch Talk can be set up for a variety of domains such as architecture related topics, testing or management related topics [Lee and Yong 2010]. A Lunch Talk exists in three forms:

- *Ad hoc*
- *Optional (weekly, monthly, quarterly)*
- *Scheduled*

5.2.7 Consequences

The following benefits of Lunch Talk are known:

- Breaking up information silos

- Less formal
- More creative setting
- Saving time through no planning in advance
- Enabling knowledge transfer

The following liabilities of Lunch Talk are known:

- Taking away important time to rest
- Quality of results during lunch might be lower
- Aversion against meetings during lunch time can make the organization difficult
- Providing right incentives to give up a lunch break is challenging

5.2.8 See Also

Lunch Talk may be applied in combination with the following M-Patterns:

- M-33: Collaborative Architecture Decision Making

5.2.9 Known Uses

The following uses of Lunch Talk are known:

- RetailCo
- SoftCo
- TechCo

5.3 V-Pattern: Architecture Solution Space (V-19) *

V-Pattern Overview	
Alias	Architecture Box, Product Solution Space
Summary	Architecture Solution Space describes architectural constraints, dependencies and boundary conditions for a large-scale software development project and is created collaboratively by experienced architects and developers.
Type	Board

5.3.1 Example

At the beginning of a software development project at RetailCo, the enterprise and solution architect have to define the necessary minimum requirements for the new software product. The latter include the ability to integrate the product seamlessly into the existing product landscape while complying to existing architectural specifications. The architects are struggling because they have to find a tradeoff between minimum requirements and at the same time not restricting the developers' freedom of decision regarding implementation.

5.3.2 Context

Due to the agile transformation, organizations have to create and design architectures differently. Especially, if the organization has just started with the transformation and the maturity is low, the field of tension between enterprise architecture management and agile methods is large. On the one hand, the teams should have maximal autonomy. On the other hand, there must be enterprise architecture management specifications in order to create a sustainable architecture without creating monolithic systems. Balancing intentional and emergent architecture becomes one of the major challenges. If there is no framework, this can lead to chaos, high costs and dissatisfaction of employees.

5.3.3 Problem

The following concerns are addressed by Architecture Solution Space:

- *C-6: How to deal with technical debts?*
- *C-25: How to manage and integrate heterogeneous subsystems of different development teams?*
- *C-34: How to ensure the reuse of enterprise assets?*
- *C-38: How to facilitate standardization across agile teams?*
- *C-69: How to verify and control the compliance of agile teams with architecture principles?*
- *C-72: How to deal with architecture-related uncertainties and risks?*
- *C-81: How to create scalable software?*

5.3.4 Forces

The following forces influence Architecture Solution Space:

- *Field of tension between enterprise architecture management and agile Methods:* Agility thrives on autonomy and freedom of decision, while enterprise architecture management depends on strict guidelines. Both contradict each other and are therefore difficult to combine, especially in large-scale agile development projects.
- *Complete overview of existing solutions is missing:* Especially in large organizations, a complete overview of already existing solutions is lacking. This is one of the reasons why the reuse of enterprise assets is difficult to ensure.
- *Agility restricts the ability for detail or to achieve certain requirements in a short amount of time:* For example, achieving scalability is only possible in a continuous process which can not be achieved to full extent in one simple sprint. Scalability affects all layers: infrastructure, domain, application and presentation layer. Therefor, relevant requirements must be defined in advance and be part of the Architecture Solution Space: which is difficult because in agile environments there is less time for planning architectures and identifying possible risks or uncertainties. Hence, not all aspects can be considered and architects have to find a tradeoff between quality and time.

5.3.5 Solution

In general, the Architecture Solution Space describes architectural constraints, dependencies and boundary conditions for a large-scale software development program and is created collaboratively by experienced architects and developers. At the beginning of the development program, it just contains minimal but essential requirements such as security constraints. As development progresses, new important elements are added. The definition of a minimal Architecture Solution Space in advance not only ensures the compliance with essential architectural standards, guidelines and dependencies but also to keep costs caused by expensive adaptions to a minimum. In addition, it decreases the field of tension between enterprise architecture management and agile methods by allowing agile teams to develop architecture intentionally without strong restrictions.

A Architecture Solution Space can contain the following elements:

- Purpose and target

- Definitions of relevant terms and abbreviations
- Architecture goals
- References to applications that can be reused
- Dependencies to other applications
- Architecture principles and guidelines

5.3.6 Variants

Architecture Solution Space can either be created by architects only or collaboratively with agile teams.

5.3.7 Consequences

The following benefits of Architecture Solution Space are known:

- Visualizes dependencies
- Provides guidance
- Helps revealing unplanned risks and dependencies in advance
- Provides a clean framework
- Helps not to restrict freedom of developers
- Reduces the costs caused by expensive adaptions or reworks

The following liabilities of Architecture Solution Space are known:

- High manual effort during its creation
- Only effective if created collaboratively by experiences architects and developers
- After the Common Planning, it might be abandoned
- Less effective if not maintained properly
- Control of adherence is difficult
- No consequences in case of non-compliance
- Strongly relies on the intrinsic motivation of developers and architects

5.3.8 Data Collection

Architecture Solution Space is created collaboratively by experienced architects and developers. At the beginning of a large-scale software development program, it just contains the minimal necessary requirements. As the development progresses,further requirements are added.

5.3.9 See Also

Architecture Solution Space is connected to the following M-Patterns:

- M-28: Principle based intentional Architecture
- M-30: Architectural Spike
- M-41: Define Architecture Principles as Quality Gate Policies

5.3.10 Known Uses

The following uses of Architecture Solution Space are known:

- CarCo
- GlobalInsureCo
- ItCo
- PublicInsureCo
- RetailCo
- SoftCo

5.4 M-Pattern: Principle Based Intentional Architecture (M-28) *

M-Pattern Overview	
Summary	Principle Based Intentional Architecture provides an agile architecture approach with a good trade off between traditional enterprise architecture practices and agile methods.
Type	Software Architecture

5.4.1 Example

At TechCo, five agile teams have to develop a large software system using agile methods. As a requirement, the system should be integratable into the existing landscape and follow the architectural direction of the company. Directly in the beginning, one enterprise architect and one solution architect use a structured approach to design an architecture target picture without involving the agile teams. During the development phase, the team struggles due to functional redundancy and has to adapt the architecture several times.

5.4.2 Context

With the application of agile methods, long planning and conception phases of traditional architecture approaches no longer exist. Whereas traditional development models such as Waterfall Model and Spiral Model impose a structured design approach leading to extensive early architecture work, agile methods force agile teams to develop architectures iteratively and perform constant software tests based on current requirements. Often, organizations struggle to balance emergent and intentional architecture because in practice, emergent architecture is rather suitable on team level but difficult to achieve on program level where several agile teams are involved. Emergent architecture design implies that architecture is designed incrementally, just as little as possible, just in time and at the last possible moment. In a big software system development project involving several agile teams, this can lead to functional redundancy, divergence of architecture and hence, increase the complexity. In order to overcome these challenges, principles and guidelines such as Loose Coupling of Systems as part of the intentional architecture design are essential.

5.4.3 Problem

The following concerns are addressed by Principle Based Intentional Architecture:

- *C-38: How to facilitate standardization across agile teams?*
- *C-68: How to deal with centralized and top-down architectural decision?*
- *C-73: How to establish a common architectural direction across the organization?*
- *C-74: How to balance intentional and emergent architecture?*
- *C-76: How to deal with the reduced time for planning architectures?*

5.4.4 Forces

The following forces influence Principle Based Intentional Architecture:

- *Field of tension between enterprise architecture management and agile Methods:* Agile methods provide autonomy and freedom of decision, while enterprise architecture management depends on direct and control. Both contradict each other and are therefore difficult to combine, especially in large-scale agile development projects.
- *Avoiding analyses paralysis:* Architectural principles close the gap between strategic goals and actual implementation decisions. By focusing on important core aspects when creating a future-oriented enterprise architecture, "analysis paralysis" can be prevented. In addition, the focus allows to carefully decide what to design and manage in a top-down approach and what to leave to emergence.
- *Local optimization and focus:* Agile software development teams tend to focus more on achieving a specific and local optimum. Thus, the overview of overarching and enterprise-wide objectives is often lost. With the influence of a structure-driven force such as enterprise architecture management, this problem can be counteracted.

5.4.5 Solution

Principle Based Intentional Architecture describes a way to create software architectures in an agile environment with the use of/based on architecture principles. In contrast to detailed architecture specifications, principles such as Build the Simplest Architecture that can Possibly work can prevent anti-patterns such as Don't Use Big Design Up Front. Furthermore principles provide guidance for developers and architects to achieve a specific future architecture design. In a large-scale agile development project, principles should be defined using a collaborative approach according to [Uludag et al. 2019c]. With the help of a Community of Practice, a tradeoff between global and local optimum can be achieved as enterprise architects and agile teams are part of the community and represent both, bottom-up and top-down perspectives [Uludag et al. 2019c].

As shown in Fig. 7, the approach consists of seven phases. *(1) Derive Drivers* describes how drivers for establishing architecture principles and guidelines should be identified. Enterprise architects should analyze sources and collect relevant inputs such as business objectives and values whereas agile teams should make use of their own implementation experience and agreements within the team. In that way, bottom-up and top-down approaches are combined providing benefits from both sides. *(2) Determine principles and guidelines* defines how principles and guidelines are created. With the help of the in *(1) Derive Drivers* identified drivers, both agile teams and enterprise architects should

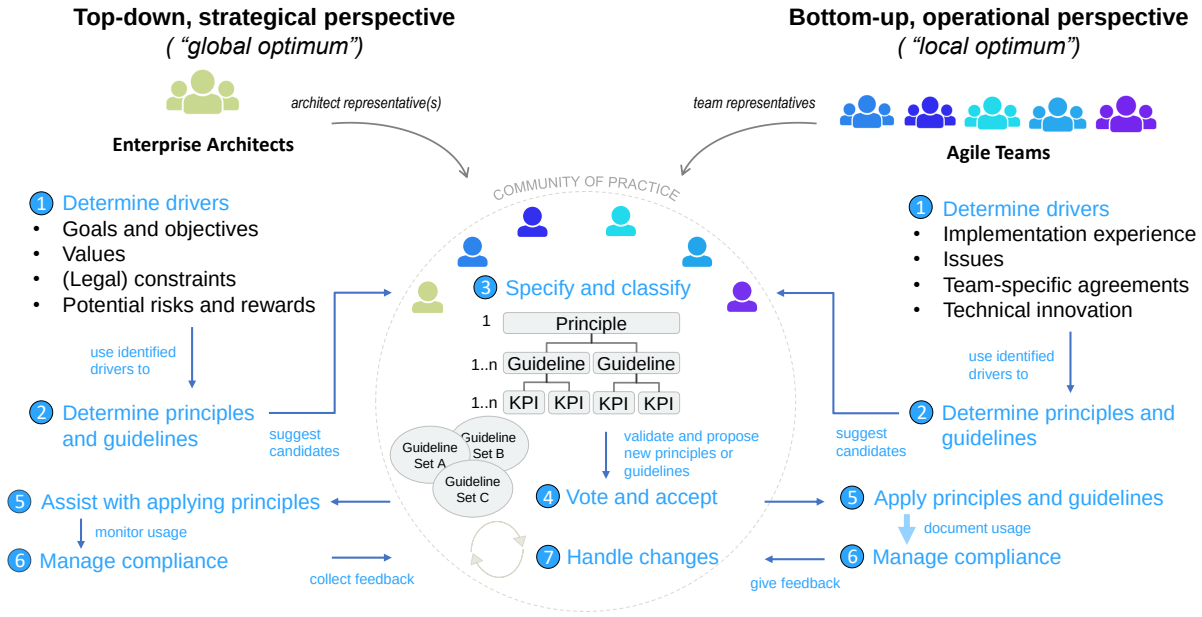

Fig. 7. Collaborative approach overview [Uludag et al. 2019c]

create a list of possible architecture principles. Afterwards relevant principles should be selected for further specification. *(3) Specify and classify* combines specification and classification of principles. In order to create principles that are most useful for a specific target group, key performance indicators should be defined as fulfillment criteria. If a principle needs to be specified even further for development, it is possible to transform it into one or more guidelines. *(4) Vote and accept* describes the process of validation. In order to establish effective principles and guidelines, a clear validation process is mandatory. With the help of a Community of Practice consisting of representatives of agile teams and enterprise architects, principles and guidelines can be discussed in detail. A democratic vote should decide, whether or not, a principle is adopted. *(5) Apply principles and guidelines* defines the application of principles and guidelines. First, enterprise architects should support agile teams during the realization of principles or guidelines that have been accepted by the community. Second, in return, agile teams should provide feedback on applied guidelines. In order to increase the intrinsic motivation of agile teams to adhere to architecture principles a gamification approach can be applied where each agile team gets a "digital" belt where a specific color indicates the success stories of compliance. *(6) Manage compliance* emphasizes how acceptance issues of agile teams to adhere to principles and guidelines can be solved. First, agile teams should be responsible to adhere to principles. Second, agile teams should be able to neglect to a specific guideline or principle. Both involves a documentation effort, as the community must be informed directly in order to be able to identify implementation or specification issues. In addition, the organization should try to perform as much automated testing as possible as manual compliance is highly time consuming. *(7) Handle changes* describes how changes should be handled. One of the most important things is to involve the experience of agile teams in the change process and provide feedback mechanisms where people can comment on specific principles, request changes or have discussions. Furthermore, enterprise architects should collect feedback from stakeholders that have an impression on the actual consequences of governance activities. All feedback should be considered by the community. Guideline or principles that only require minor changes should be adapted whereas bigger changes should lead to the creation of new ones.

5.4.6 *Variants*

A variant of Principle Based Intentional Architecture is that the definition of architectural principles is only done by architects, rather than in collaboration with agile teams.

5.4.7 *Consequences*

The following benefits of Principle Based Intentional Architecture are known:

- Provides guidance for development to achieve a specific future architecture design
- Facilitates creation of intentional architecture
- Provide solution space enabling agile teams to develop software within guideline while not restricting freedom of decision
- Prevent big design up front with applying the principle *P-24: Build the Simplest Architecture that can Possibly Work*

The following liabilities of Principle Based Intentional Architecture are known:

- Lack of control mechanisms to ensure compliance of architecture principles
- No adherence to architecture principles due to resistance of agile teams
- Lack of assessment criteria for defining architecture principles
- Requires support and enablement through enterprise architects
- Lack of feedback mechanisms during implementation

5.4.8 See Also

Principle Based Intentional Architecture is related to the following M-Patterns:

- M-22: Agile Architecture Governance Approach
- M-26: Add Principles to Definition of Done
- M-41: Define Architecture Principles as Quality Gate Policies

Additionally Principle Based Intentional Architecture is closely linked to the V-Pattern:

- V-19: Architecture Solution Space

5.4.9 Other Standards

[Fischer et al. 2010] emphasizes to use architecture principles as part of a lightweight framework to support large-scale agile software development.

5.4.10 Known Uses

The following uses of Principle Based Intentional Architecture are known:

- CarCo
- GlobalInsureCo
- ItCo
- PublicInsureCo
- RetailCo
- SoftCo
- TechCo

5.5 Anti-Pattern: Don't be a PowerPoint Architect (A-12) *

Anti-Pattern Overview	
Alias	Architects don't code, Architects Play Golf
Summary	Don't be a PowerPoint Architect shows why providing high level architecture artifacts is causing problems in an agile environment.

5.5.1 Example

Three agile teams from CarCo want to start developing a new product and need technical support to create the architecture. The specification of requirements by the product owner includes the application of a new technology. In addition, since the application has to execute crucial payment operations, the team must identify as many architecture-related uncertainties and risks as possible in advance. As only little expertise is available within the team, an enterprise architect is contacted to clarify the necessary technical concerns. The architect cannot answer the questions and only provides some high level artifacts such as diagrams, UML models and a general Word document with around 300 pages. After further inquiry, the development joint determines that the enterprise architect does not have the necessary technical know-how to answer the questions.

5.5.2 Context

In traditional software development projects architects were involved in long conception and planning phases and mainly created high level artifacts such as system diagrams and data models. Furthermore, software was mainly developed using linear sequential models, where each phase depends on the deliverables of the previous one. In the course of agile transformation, the requirements have changed and hence the working methodology. Conception and planning phases are short and development teams have to handle changing requirements in a short amount of time. Therefor it is necessary, that architects provide detailed technical guidance instead of high level abstract artifacts which do not provide sufficient support [Kulak and Li 2017].

5.5.3 Problem

The following concern is addressed by Don't be a PowerPoint Architect:

- *C-86: How to deal with the new working methodology as an architect within agile environments?*

5.5.4 Forces

The following forces occur in the context of Don't be a PowerPoint Architect:

- *Agile environment changes working methodology of architects:* With the application of agile methods, long conception and planning phases are obsolete and due to constantly changing requirements, software architectures often have to be adapted within a short amount of time which requires architects and developers to work closely together. Especially architects in traditional organizations are affected as they are specialized rather in rough planning and less in technical implementation. Furthermore, the change of working methodology incorporates a shift from indirect communication to direct communication.
- *Architects need to have a deep technical understanding:* With the approach of new technologies and development cycles getting shorter, agile teams have to develop software faster and often use new technologies. Therefor, architects need to be able to provide detailed technical support and to be able to identify architecture related risks and hotspots in advance.

5.5.5 General Form

After the success of a difficult large-scale agile development project, the organization was curious to identify the reasons for success. A survey among agile teams involved in that development project showed that especially the support of the enterprise architect played a major role. Not only did he have profound knowledge about the technologies but also provided detailed architecture related descriptions and code snippets. Thanks to the technical detail of the artifacts provided by the enterprise architect, the agile team had no difficulties during development. As a result, the organization believes, that all upcoming development projects should have an enterprise architect with the above mentioned characteristics to ensure project success.

5.5.6 Consequences

The following benefits of Don't be a PowerPoint Architect are known:

- Decreases communication gap between architect and agile team
- Increases acceptance of agile teams
- Increases understanding of architecture
- Increases intrinsic motivation to adhere to architecture principles

The following liabilities of Don't be a PowerPoint Architect are known:

- Support and enabling takes a lot of time
- Architects need to have enough capacity
- Requires a broad and deep skill set which is rare

5.5.7 *Revised Solution*

The most important aspect of the revised solution is: Try to avoid Don't be a PowerPoint Architect. This can be achieved by a few steps:

- *Coaching and mentoring:* To ensure a sufficient support with detailed technical artifacts, organizations need to provide coaching and mentoring for enterprise architects.
- *Ensure sufficient capacity:* To solve the capacity problem, first, organizations must adapt accordingly and increase the number of enterprise architects. Second, a phase wise supporting approach should be used to help as many agile teams as possible.
- *Lack of feedback:* Often, organizations do not implement proper feedback mechanisms and/or make them a mandatory requirement. Therefore, agile teams are not able to provide proper feedback about the work of enterprise architects. Hence, the misconception of ineffective artifacts provided by enterprise architects can not be uncovered.
- *Change hiring strategy:* Support and enablement require new skills such as deep technical know-how of old and especially new technologies and social skills to overcome the acceptance problem of enterprise architects by agile teams. Therefore, when hiring enterprise architects, these characteristics should be mandatory.

5.5.8 *Known Uses*

The following uses of Don't be a PowerPoint Architect are known:

- CarCo
- ItCo
- PublicInsureCo
- RetailCo
- TechCo

5.5.9 *See Also*

The anti-pattern Don't be a PowerPoint Architect is strongly connected to the M-Pattern *Supporting Architect.*

5.5.10 *Other standards*

[Kulak and Li 2017; Subramaniam and Hunt 2006] emphasize that architects must be able to write code in order to provide sufficient support.

> *"They typically come in during the beginning of a project, draw all kinds of diagrams, and leave before any serious implementation takes place. There are many "PowerPoint architects" out there, and they aren't effective because of lack of feedback."*

According to [Subramaniam and Hunt 2006], establishing feedback mechanisms for developers can help overcome this problem.

6 Discussion

In the following, the main key findings are discussed.

Increase of agile methods in large-scale development projects leads to Anti-Patterns/changes working methodology: Most of the interviewed organizations adopted agile methods such as Scrum and XP for developing large software systems. Since the agile development methodology focuses on being adaptive rather than predictive organizations had to change traditional ways of developing software such as Big Design Up Front. We observed that especially traditional enterprise architects in large organizations, are used to long conception and planning phases where detailed architectures were planned. According to the interviewees, nowadays architectures have to be developed within a short amount of time and less detail as it can delay the development of the software (see Anti-Pattern Don't Use Big Design Up Front).
While in some organizations the distance between enterprise architects and agile teams has decreased significantly, there are still some outliers. Some of the agile teams complained about enterprise architects that provide too complex architecture models with a wrong level of abstraction and hence, no impact. In line with [Van Der Raadt et al. 2008] this situation is referred to as the "ivory tower" syndrome. Similar to the anti-pattern "modeling for the modeling's sake" defined by [Ambler 2008] we identified, that enterprise architects should be developing architectures in close collaboration with agile teams (see Anti-Pattern Don't Build an Ivory Tower) and use direct communication in order to provide sufficient support and prevent misunderstandings (see Anti-Pattern Don't use Indirect Communication). Furthermore, we noticed that a lot of agile teams and solution architects are tired of high-level diagrams and architecture models provided via PowerPoint since the lack of detail hinders the effectiveness. In accordance with [Kulak and Li 2017] we discovered that architects should be able to write code on a regular basis in order to provide sufficient technical support to agile teams (Anti-Pattern Donŕt be a PowerPoint architect).

Role of the supporting architect is getting more and more popular: Similar to [Drews et al. 2017] we observe that the working methodology of enterprise architects among more and more organizations have changed from direct and control to support and enablement within the agile transformation. In particular, because long planning and conception phases have disappeared due to the use of agile methods. As described in the M-Pattern Supporting Architect, based on our results, enterprise architect must still be able to maintain an enterprise-wide focus. This is particularly important for large-scale agile development projects, as agile teams can also gain an understanding of the entire architecture. Furthermore, we have discovered that especially the collaboration between enterprise architects and agile teams is increasingly important in order to ensure development speed or solve architectural challenges in a short amount of time. Therefore, enterprise architects need to have different characteristics such as the ability to code for short demonstration, provide delivery pipelines and architectural spikes to reduce architectural uncertainties and risks [Kulak and Li 2017; Uludag et al. 2019b]. Furthermore social skills are necessary to learn about the developers and their strength and weaknesses which is important in order to increase the intrinsic motivation to adhere to architecture principles.

Architecture solution space helps to balance intentional and emergent architecture: According to the interviewed organizations finding the balance between emergent and intentional architecture is one of the major challenges in large-scale agile development projects [Uludağ et al. 2017; Waterman 2014] as emergent architecture is rather suitable on team level but difficult to achieve on program level where several agile teams are involved. We experience, that although most of the agile teams try to follow the emergent design principle of the Agile Manifesto [Beck et al. 2001] a lot of developers want to be part of the creation of an intentional architecture. The problem at hand originates from the application of agile methods. The latter imply that an architecture should evolve incrementally rather than forced through a direct structuring force. In a large-scale agile software development project, this can lead to redundancy, increase of complexity of architecture and huge costs through redesign efforts. Similar to [Uludag et al. 2019c] our results indicate that balancing intentional and emergent architecture requires not only a close collaboration between agile teams and architects but also some architectural guidance. With the help of the V-Pattern Architecture Solution Space this balance can be achieved as it allows agile teams to move within a certain space and intentionally develop architectures without strong restrictions.

Compliance to architecture related activities is difficult: To date, the interviewed organizations have hardly found any solutions for compliance to architecture related activities. This is also due to the fact that, for example, monitoring agile teams in a scaled environment is becoming increasingly difficult. For this reason, organizations must consider other methods or solutions such as the M-Pattern Architectural Thinking [Winter 2014]. Note that this should not be confused with classical enterprise architecture management because architectural thinking promotes the individual decision maker within the organization to take responsibility [Aier et al. 2015]. In order to facilitate architectural thinking successfully, we emphasize to establish an architectural thinking in an organization similar to [Weiss et al. 2013; Winter 2014]. Furthermore, the focus should be on both, long-term and short-term goals. In accordance with [Lattanze 2011; Ross and Quaadgras 2012] organizations should offer enterprise architecture management training programs to help establish the long-term goal of an architectural perspective in the entire organization. It has to be considered, however, that training should not only promote the understanding of architecture but also anchor the importance of architecture firmly in the mindset. Another possibility would be to use the M-Pattern Architecture Governance through Institutional Pressure which was highlighted by some of the interviewees. They argued, that compliance can only be achieved if employees understand the benefits of adherence. This is consistent with the [Uludag et al. 2019c] approach, which recommends an enhanced collaborative approach. A gamification approach is used to ensure the necessary intrinsic motivation. On the one hand, awards for compliance exert a normative pressure on agile teams. On the other hand, through transparency, other

agile teams experience mimetic pressures. With this approach it is possible to persuade agile teams to check and motivate themselves [Uludag et al. 2019c].

7 Conclusion and Outlook

Due to the success of agile methods in small, autonomous and independent teams, companies try to make use of these benefits in large-scale agile software development projects while building complex software systems [Alqudah and Razali 2016; Dingsøyr et al. 2014]. Given the heterogeneity of large-scale projects and the changing state of methods and frameworks, this area is particularly prone to practice-oriented design research. Furthermore, the scaling of agile methods is accompanied by many architectural challenges such as managing dependencies of different systems and balancing intentional and emergent architecture [Dikert et al. 2016; Dingsøyr et al. 2019; Uludag et al. 2018]. Especially enterprise architects and solution architects are confronted with some unprecedented challenges [Uludag et al. 2018]. Despite the importance of this area for the success of large-scale agile development projects, the literature lacks both a description of challenges and best practices.

To close this research gap, the proposed model language was used. The latter provides a structure for documenting challenges and best practices and includes concerns, principles, coordination patterns, methodology patterns, viewpoint patterns and anti-patterns of typical stakeholders in the agile environment. With the help of a case study and expert interviews, 19 concerns and 43 patterns were identified. Five of the latter were presented in this paper.

This paper leaves room for future research. Since the pattern language is based on the *Rule of Three* to distinguish between pattern candidates and actual patterns, additional quantitative studies should be conducted in other organizations to generalize and validate the results, identify further concerns and best practices and measure the incidence of various views and opinions.

References

ISO/IEC/IEEE 42010:2011(E). 2011. *Systems and software engineering – Architecture description.* Technical Report. ISO/IEC/IEEE.

Stephan Aier, Nils Labusch, and Patrick Pähler. 2015. Implementing architectural thinking. In *International Conference on Advanced Information Systems Engineering*. Springer, 389–400.

Christopher Alexander. 1977. *A Pattern Language: Towns, Buildings, Construction.* Oxford University Press, New York.

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

Wasim Alsaqaf, Maya Daneva, and Roel Wieringa. 2019. Quality requirements challenges in the context of large-scale distributed agile: An empirical study. *Information and software technology* 110 (2019), 39–55.

Scott Ambler. 2008. Enterprise Modeling Anti-Patterns. Agile Modeling. http://www.agilemodeling.com/essays/enterpriseModelingAntiPatterns.htm. Accessed: 2019-09-01.

Scott W Ambler. 1998. *Process patterns: building large-scale systems using object technology.* Cambridge University Press.

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

Kent Beck and Erich Gamma. 2000. *Extreme programming explained: embrace change.* addison-wesley professional.

Mike Beedle, James O. Coplien, Jeff Sutherland, Jens C. Østergaard, Ademar Aguiar, and Ken Schwaber. 2010. Essential Scrum Patterns. In *14th European Conference on Pattern Languages of Programs*. The Hillside Group, Irsee, 1–17.

Mike Beedle, Martine Devos, Yonat Sharon, Ken Schwaber, and Jeff Sutherland. 1999. SCRUM: An Extension Pattern Language for Hyperproductive Software Development. *Pattern Languages of Program Design* 4 (1999), 637–651.

Patrick Besson and Frantz Rowe. 2012. Strategizing information systems-enabled organizational transformation: A transdisciplinary review and new directions. *The Journal of Strategic Information Systems* 21, 2 (2012), 103–124.

Barry Boehm and Richard Turner. 2005a. Management challenges to implementing agile processes in traditional development organizations. *IEEE software* 22, 5 (2005), 30–39.

Barry W Boehm. 1988. A spiral model of software development and enhancement. *Computer* 5 (1988), 61–72.

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

Alexander Bogner, Beate Littig, and Wolfgang Menz. 2009. Introduction: Expert interviews. An introduction to a new methodological debate. In *Interviewing experts*. Springer, 1–13.

Teodora Bozheva and Maria Elisa Gallo. 2005. Framework of agile patterns. In *European Conference on Software Process Improvement*. Springer, 4–15.

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

Frank Buschmann, Kevlin Henney, and C. Schmidt Douglas. 2007a. *Pattern Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing.* John Wiley & Sons, Chichester.

Frank Buschmann, Kevlin Henney, and C. Schmidt Douglas. 2007b. *Pattern Oriented Software Architecture Volume 5: On Patterns and Pattern Languages.* John Wiley & Sons, Chichester.

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. 1996. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns.* John Wiley & Sons, Chichester.

James O. Coplien. 1995. A Generative Development-process Pattern Language. In *Pattern Languages of Program Design*, James O. Coplien and Douglas C. Schmidt (Eds.). ACM, New York, 183–237.

James O. Coplien. 1996. *Software Patterns: Management Briefs.* Cambridge university Press, Cambridge.

James O. Coplien and Neil B. Harrison. 2004. *Organizational Patterns of Agile Software Development.* Addison-Wesley, Boston.

Kim Dikert, Maria Paasivaara, and Casper Lassenius. 2016. Challenges and Success Factors for Large-Scale Agile Transformations: A Systematic Literature Review. *Journal of Systems and Software* 119 (2016), 87–108.

Torgeir Dingsøyr, Davide Falessi, and Ken Power. 2019. Agile Development at Scale: The Next Frontier. *IEEE Software* (2019). Special Issue: Large-Scale Agile Development.

Torgeir Dingsøyr and Nils Moe. 2014a. *Towards Principles of Large-Scale Agile Development.* Springer, Berlin, 1–8.

Torgeir Dingsøyr and Nils Brede Moe. 2014b. Towards principles of large-scale agile development. In *International Conference on Agile Software Development*. Springer, 1–8.

Torgeir Dingsøyr, Nils Brede Moe, Roberto Tonelli, Steve Counsell, Cigdem Gencel, and Kai Petersen. 2014. *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation: XP 2014 International Workshops, Rome, Italy, May 26-30, 2014, Revised Selected Papers.* Vol. 199. Springer.

Paul Drews, Ingrid Schirmer, Bettina Horlach, and Carsten Tekaat. 2017. Bimodal enterprise architecture management: The emergence of a New EAM function for a BizDevOps-based fast IT. In *2017 IEEE 21st International Enterprise Distributed Object Computing Workshop (EDOCW)*. IEEE, 57–64.

Tore Dyba and Torgeir Dingsoyr. 2009. What do we know about agile software development? *IEEE software* 26, 5 (2009), 6–9.

Amr Elssamadisy. 2007. *Patterns of Agile Practice Adoption.* Lulu. com.

Amr Elssamadisy. 2008. *Agile Adoption Patterns: A Roadmap to Organizational Success.* Addison-Wesley, Boston.

Alexander M. Ernst. 2010. *A Pattern-based Approach to Enterprise Architecture Management.* Dissertation. Technische Universität München, München.

Christian Fischer, Robert Winter, and Stephan Aier. 2010. What is an enterprise architecture principle? In *Computer and Information Science 2010*. Springer, 193–205.

Martin Fowler. 2006. Writing Software Patterns. https://www.martinfowler.com/articles/writingPatterns.html. Accessed: 2019-02-02.

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

Daniel Gerster, Christian Dremel, and Prashant Kelker. 2018. " Agile meets non-agile": Implications of adopting agile practices at enterprises. (2018).

Neil B. Harrison. 1996. Organizational Patterns for Teams. In *Pattern Languages of Program Design 2*, John M. Vlissides, James O. Coplien, and Norman L. Kerth (Eds.). Addison-Wesley, Boston, 345–352.

Muhammad Atif Javed, Srdjan Stevanetic, and Uwe Zdun. 2016. Towards a pattern language for construction and maintenance of software architecture traceability links. In *Proceedings of the 21st European Conference on Pattern Languages of Programs*. ACM, 24.

Martin Kalenda, Petr Hyna, and Bruno Rossi. 2018. Scaling agile in large organizations: Practices, challenges, and success factors. *Journal of Software: Evolution and Process* 30, 10 (2018), e1954.

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

Petri Kettunen. 2007. Extending Software Project Agility with new Product Development Enterprise Agility. *Software Process: Improvement and Practice* 12, 6 (2007), 541–548.

Petri Kettunen and Maarit Laanti. 2008. Combining agile software projects and large-scale organizational agility. *Software Process: Improvement and Practice* 13, 2 (2008), 183–193.

Kurt Keutzer, Berna L Massingill, Timothy G Mattson, and Beverly A Sanders. 2010. A design pattern language for engineering (parallel) software: merging the PLPP and OPL projects. In *Proceedings of the 2010 Workshop on Parallel Programming Patterns*. ACM New York, NY, USA, 1–8.

Daryl Kulak and Hong Li. 2017. *The Journey to Enterprise Agility: Systems Thinking and Organizational Legacy*. Springer.

Craig Larman and Victor R Basili. 2003. Iterative and incremental developments. a brief history. *Computer* 36, 6 (2003), 47–56.

Anthony J Lattanze. 2011. Infusing architectural thinking into organizations. *IEEE software* 29, 1 (2011), 19–22.

Seiyoung Lee and Hwan-Seung Yong. 2010. Distributed agile: project management in a global environment. *Empirical Software Engineering* 15, 2 (2010), 204–217.

Dean Leffingwell. 2007. *Scaling software agility: best practices for large enterprises*. Pearson Education.

Christian Lescher. 2010. Patterns for global development: how to build one global team?. In *Proceedings of the 15th European Conference on Pattern Languages of Programs*. ACM, 6.

Neil Maiden and Sara Jones. 2010. Agile Requirements Can We Have Our Cake and Eat It Too? *IEEE Software* 27, 3 (2010), 87–88.

Ian Mitchell. 2016. *Agile Development in Practice*. TamaRe House, London.

Miguel J Monasor, Aurora Vizcaíno, Mario Piattini, John Noll, and Sarah Beecham. 2013. Towards a global software development community web: Identifying patterns and scenarios. In *2013 IEEE 8th International Conference on Global Software Engineering Workshops*. IEEE, 41–46.

Maria Paasivaara and Casper Lassenius. 2014. Communities of practice in a large distributed agile software development organization âĂŞ Case Ericsson. *Information and Software Technology* 56, 12 (2014), 1556 – 1577. Special issue: Human Factors in Software Development.

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

JW Ross and A Quaadgras. 2012. Enterprise architecture is not just for architects. *CISR Research Briefings* 7, 9 (2012).

Jeanne W Ross, Peter Weill, and David Robertson. 2006. *Enterprise architecture as strategy: Creating a foundation for business execution*. Harvard Business Press.

Dominik Rost, Balthasar Weitzel, Matthias Naab, Torsten Lenhart, and Hartmut Schmitt. 2015. Distilling best practices for agile development from architecture methodology. In *European Conference on Software Architecture*. Springer, 259–267.

Winston W Royce. 1987. Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering*. IEEE Computer Society Press, 328–338.

Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering* 14, 2 (2009), 131.

Nayan B Ruparelia. 2010. Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes* 35, 3 (2010), 8–13.

Fabiano B Ruy, Ricardo A Falbo, Monalessa P Barcellos, Giancarlo Guizzardi, and Glaice KS Quirino. 2015. An ISO-based software process ontology pattern language and its application for harmonizing standards. *ACM SIGAPP Applied Computing Review* 15, 2 (2015), 27–40.

Ken Schwaber. 1997. Scrum development process. In *Business object design and implementation*. Springer, 117–134.

Ken Schwaber and Mike Beedle. 2002. *Agile software development with Scrum*. Vol. 1. Prentice Hall Upper Saddle River.

ScrumPLoP. 2019. Published Patterns. https://sites.google.com/a/scrumplop.org/published-patterns/. Accessed: 2019-08-17.

Venkat Subramaniam and Andy Hunt. 2006. *Practices of an agile developer: Working in the real world*. Pragmatic Bookshelf.

Jeff Sutherland, Neil Harrison, and Joel Riddle. 2014. Teams that finish early accelerate faster: a pattern language for high performing scrum teams. In *2014 47th Hawaii International Conference on System Sciences*. IEEE, 4722–4728.

Abbas Tashakkori and Charles Teddlie. 2010. *Sage handbook of mixed methods in social & behavioral research*. sage.

Paul Taylor. 2000. Capable, productive, and satisfied: Some organizational patterns for protecting productive people. In *Pattern Languages of Program Design 4*, John M. Vlissides, James O. Coplien, and Norman L. Kerth (Eds.). Addison-Wesley, Boston, 611–636.

Ömer Uludag, Nina Harders, and Florian Matthes. 2019a. Documenting Recurring Concerns and Patterns in Large-Scale Agile Development.. In *In 24th European Conference on Pattern Languages of Programs*.

Ömer Uludağ, Matheus Hauder, Martin Kleehaus, Christina Schimpfle, and Florian Matthes. 2018. Supporting large-scale agile development with domain-driven design. In *International Conference on Agile Software Development*. Springer, 232–247.

Ömer Uludag, Martin Kleehaus, Christoph Caprano, and Florian Matthes. 2018. Identifying and structuring challenges in large-scale agile development based on a structured literature review. In *2018 IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 191–197.

Ömer Uludag, Martin Kleehaus, Niklas Reiter, and Florian Matthes. 2019b. What to Expect from Enterprise Architects in Large-Scale Agile Development? A Multiple-Case Study. (2019).

Ömer Uludağ, Martin Kleehaus, Xian Xu, and Florian Matthes. 2017. Investigating the role of architects in scaling agile frameworks. In *2017 IEEE 21st International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 123–132.

Ömer Uludag, Sascha Nägele, and Matheus Hauder. 2019c. Establishing Architecture Guidelines in Large-Scale Agile Development Through Institutional Pressures: A Single-Case Study. (2019).

Ömer Uludağ, Martin Kleehaus, Christoph Caprano, and Florian Matthes. 2018. Identifying and Structuring Challenges in Large-Scale Agile Development Based on a Structured Literature Review. In *22nd International Enterprise Distributed Object Computing Conference*. IEEE, Stockholm, 191–197.

Antti Välimäki. 2011. *Pattern Language for Project Management in Global Software Development*. Tampere University of Technology, Tampere.

Bas Van Der Raadt, Sander Schouten, and Hans Van Vliet. 2008. Stakeholder perception of enterprise architecture. In *European Conference on Software Architecture*. Springer, 19–34.

VersionOne. 2018. *12th Annual State of Agile Report*. Technical Report. VersionOne.

VersionOne. 2019. VersionOne Inc.13th annual state of agile report. https://www.stateofagile.com/. Accessed: 2019-08-15.

Jan Vom Brocke, Alexander Simons, Bjoern Niehaves, Kai Riemer, Ralf Plattfaut, Anne Cleven, et al. 2009. Reconstructing the giant: on the importance of rigour in documenting the literature search process.. In *Ecis*, Vol. 9. 2206–2217.

Michael Grant Waterman. 2014. Reconciling agility and architecture: a theory of agile architecture. (2014).

Simon Weiss, Stephan Aier, and Robert Winter. 2013. Institutionalization and the effectiveness of enterprise architecture management. Association for Information Systems.

Robert Winter. 2014. Architectural thinking. *Business & Information Systems Engineering* 6, 6 (2014), 361–364.

Robert K Yin. 2017. *Case study research and applications: Design and methods*. Sage publications.

A Pattern Language for Enterprise and Solution Architects

A.1 Stakeholders

- **S-1:** Development Team
- **S-2:** Product Owner
- **S-3:** Scrum Master
- **S-4:** Software Architect
- **S-5:** Test Team
- **S-6:** Product Manager
- **S-7:** Program Manager
- **S-8:** Agile Coach
- **S-9:** Enterprise Architect
- **S-10:** Business Analyst
- **S-11:** Solution Architect
- **S-12:** Portfolio Manager
- **S-13:** Support Engineer
- **S-14:** UX Expert

A.2 Concerns

- **C-2:** *How to consider integration issues and dependencies with other teams?*
- **C-6:** *How to deal with technical debts?*
- **C-8:** *How to ensure that non functional requirements are considered by the development team?*
- **C-9:** *How to find the right balance between architectural improvements and business value?*
- **C-14:** *How to create a proper upfront architecture design of the system?*
- **C-21:** *How to manage dependencies to other existing environments?*
- **C-25:** *How to manage and integrate heterogeneous subsystems of different development teams?*
- **C-26:** *How to align and communicate architectural decisions?*
- **C-27:** *How to manage and share knowledge about system components and dependencies?*
- **C-34:** *How to ensure the reuse of enterprise assets?*
- **C-38:** *How to facilitate standardization across agile teams?*
- **C-40:** *How to apply agile practices for developing/maintaining legacy systems?*
- **C-51:** *How to ensure that agile teams adhere to architecture-related activities?*
- **C-57:** *How to decompose agile teams in smaller independent teams?*
- **C-64:** *How to define a lightweight review process for adopting new technology?*
- **C-67:** *How to deal with a lack of understanding of architecture?*
- **C-68:** *How to deal with centralized and top-down architectural decision?*
- **C-69:** *How to verify and control the compliance of ATs with architecture principles?*
- **C-70:** *How to assign systems to business domains?*
- **C-71:** *How to decompose monolithic systems?*
- **C-72:** *How to deal with architecture-related uncertainties and risks?*
- **C-73:** *How to establish a common architectural direction across the organization?*
- **C-74:** *How to balance intentional and emergent architecture?*
- **C-75:** *How to deal with communication gaps between EA and AT?*
- **C-76:** *How to deal with the reduced time for planning architectures?*
- **C-77:** *How to identify hotspots within the architecture?*
- **C-78:** *How to ensure that architecture check-ins are controlled?*
- **C-79:** *How to decide whether to make or buy?*
- **C-80:** *How to align business and IT?*
- **C-81:** *How to create scalable software?*
- **C-82:** *How to integrate internal and external cloud?*
- **C-83:** *How to develop software systems that are open for third parties?*
- **C-84:** *How to migrate applications to the cloud?*
- **C-85:** *How to align enterprise architecture and product management?*
- **C-86:** *How to deal with the new working methodology as an architect within agile environments?*

A.3 Anti-Patterns

- **A-11:** Don't Use Best of Breed
- **A-12:** Don't be a PowerPoint Architect
- **A-13:** Don't Use Big Design Up Front
- **A-14:** Don't Separate between Enterprise and Software Architecture
- **A-15:** Don't Build an Ivory Tower
- **A-16:** Don't Overshoot Coordination Meetings
- **A-17:** Don't Use Indirect Communication
- **A-18:** Don't Force Requirements Without Support
- **A-19:** Don't Create Data Silos

A.4 Principles

- **P-18:** Collocate Architects with Agile Teams
- **P-19:** IT Systems Communicate through Services
- **P-20:** End to End Responsibility
- **P-21:** Loose Coupling of Systems
- **P-22:** Reuse is Preferable to Buy, which is Preferable to Make
- **P-23:** Applications rely on One Technology Stack
- **P-24:** Build the Simplest Architecture that Can Possibly Work
- **P-25:** Use Microservices
- **P-26:** API First
- **P-27:** Cloud First
- **P-28:** Use Direct Communication
- **P-29:** Develop Competition Critical Software Systems Inhouse
- **P-30:** Strictly Separate Build and Run Stages
- **P-31:** Composition Over Inheritance
- **P-32:** IT Systems Communicate through Services

A.5 M-Patterns

- **M-22:** Agile Architecture Governance Approach
- **M-23:** Enterprise Architecture Governance Service
- **M-24:** Architecture Governance through Institutional Pressure
- **M-25:** Architectural Thinking
- **M-26:** Add Principles to DoD
- **M-27:** Quality Gate
- **M-28:** Principle based Intentional Architecture
- **M-29:** Supporting Architect
- **M-30:** Architectural Spikes
- **M-31:** Agile Collaboration Environment
- **M-32:** Architecture Gate
- **M-33:** Collaborative Architecture Decision Making
- **M-34:** Architectural Runway
- **M-35:** Empowered Community of Practice
- **M-36:** Plan Additional Time for Enablers
- **M-37:** Domain Driven Design
- **M-38:** Business Capability Centric Teams
- **M-39:** Collaborative Adoption of new Technologies

- **M-40:** Defining Architecture Principles as Quality Gate Policies

A.6 CO-Patterns

- **CO-1:** Community of Practice
- **CO-15:** Center of Excellence
- **CO-16:** Lunch Talk

A.7 V-Patterns

- **V-19:** Architecture Solution Space
- **V-20:** Business Domain Map
- **V-21:** Business Capability Map
- **V-22:** Technical Debt Backlog
- **V-23:** Communication Diagram
- **V-24:** Context Map
- **V-25:** Data Diagram
- **V-26:** Weighted Shortest Job First
- **V-27:** Number of API Calls
- **V-28:** Number of Changes of Architecture Models
- **V-29:** Number of Consulting Requests
- **V-30:** Number of Dependencies
- **V-31:** Number of Releases
- **V-32:** Number of Version Skippings
- **V-33:** System Diagram
- **V-34:** Time to Feature Delivery
- **V-35:** Total Cost of Ownership
- **V-36:** Technology Radar