Documenting Recurring Concerns and Patterns in Large-Scale Agile Development

Ömer Uludağ
Technische Universität München
Garching bei München, Germany
oemer.uludag@tum.de

Nina-Mareike Harders
Technische Universität München
Garching bei München, Germany
nina.harders@tum.de

Florian Matthes
Technische Universität München
Garching bei München, Germany
matthes@tum.de

Abstract

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

CCS Concepts

• **Software and its engineering → Agile software development**.

Keywords

concerns, large-scale agile development, patterns

ACM Reference Format:
Ömer Uludağ, Nina-Mareike Harders, and Florian Matthes. 2019. Documenting Recurring Concerns and Patterns in Large-Scale Agile Development. In *Proceedings of ACM Conference (EuroPLoP'19).* ACM, New York, NY, USA, 17 pages. https://doi.org/0000001.0000001_2

1 Introduction

Over the past two decades, software development has experienced substantial growth in the use of agile methods [Maiden and Jones 2010]. Unlike traditional methods that focus on upfront plans and documentation, agile methods such as Extreme Programming and Scrum strongly encourage team collaboration, change tolerance, evolutionary delivery, and active customer involvement [Dingsøyr and Moe 2014; Kettunen 2007]. The fundamental assumptions are that small, self-organizing teams develop adaptive software using the principles of continuous design improvement and testing based on rapid iterations and frequent feedback loops [Nerur et al. 2005]. Hitherto, agile methods were mostly applied within the so-called "agile sweet spot": small, co-located teams of less than 15 people doing greenfield development for non-safety-critical systems in a volatile environment [Kruchten 2013; Nord et al. 2014]. Their proven and potential benefits made them also attractive for projects outside of the sweet spot [Dikert et al. 2016]. Thus, there is an industry trend towards introducing agile methods at scale [Dikert et al. 2016; VersionOne 2018]. The term 'large-scale agile development' is used to describe multi-team development efforts that make use of agile principles involving a high number of actors and interfaces with existing systems [Dingsøyr and Moe 2014; Rolland et al. 2016]. However, the adoption of agile methods at larger scale entails organizations with unprecedented challenges such as general resistance to change, coordination challenges in multi-team environments, and dependencies to other existing environments [Dikert et al. 2016; Uludağ et al. 2018]. Compared to the rich body of agile software development literature describing typical challenges (cf. [Hossain et al. 2009; Inayat et al. 2015]) and best practices (cf. [Beedle et al. 2010, 1999; Coplien and Harrison 2004; ScrumPLoP 2019]), the documentation of concerns and patterns in large-scale agile development is still scarce. Our study is inspired by the pattern-based approach to Enterprise Architecture Management (EAM) [Ernst 2010; Schneider and Matthes 2015] and aims to fill this gap by providing best practices for recurring concerns of stakeholders in large-scale endeavors. As a starting point, we introduce the concept of large-scale agile development patterns and present four patterns that exemplarily demonstrate the proposed language.

The remainder of this paper is structured as follows. In Section 2, we present the research approach that follows the pattern-based design research (PDR) method. In Section 3, we provide an overview of related works in the field of large-scale agile development and describe related pattern languages. We elaborate the proposed large-scale agile development pattern language in Section 4. In Section 5, we present four patterns. Section 6 shows corresponding evaluation results. In Section 7, we conclude our paper with a summary of our results and remarks on future research.

2 Research Approach

Our research initiative aims to document best practices that address concerns in large-scale agile development. To balance the rigor and relevance of the research, we followed the pattern-based design research (PDR) method as recommended by [Buckl et al. 2013]. The PDR method enables researchers to theorize and learn from the intervention at the industry partners while performing rigorous and relevant design science research. It builds on established concepts such as patterns and design theories. As depicted in Fig. 1, the PDR method consists of four phases: *observe & conceptualize*, *pattern-based theory building & nexus instantiation*, *solution design & application*, and *evaluation & learning*. Within the *observe & conceptualize* phase, good practices for recurring concerns are

EuroPLoP'19, July 2019, Kloster Irsee, Bavaria, Germany

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/0000001.0000001_2

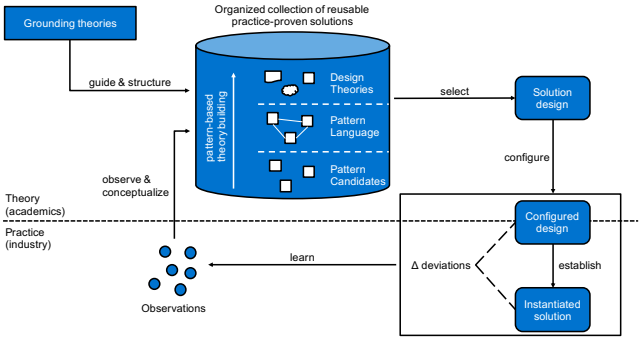

Figure 1: Pattern-based design research [Buckl et al. 2013]

observed and documented following a typical pattern structure (see Section 4). These pattern candidates are then conceptualized by using grounding theories and evolve into actual patterns by fulfilling the *rule of three*[1] [Coplien 1996], which are then integrated into the large-scale agile development pattern language. Design theories can be developed by documenting appropriate context and problem descriptions. Pattern candidates, patterns, the pattern language, and design theories together form an organized collection of reusable, proven solutions. Within the *solution design & application* phase, typical stakeholders in large-scale agile development use this knowledge base and select patterns based on their individual concerns. The selected pattern must be configured and adapted to the terminology of the company. After its configuration, the pattern can be established within the case company. During the *evaluation & learning* phase, deviations between the actual and original pattern configuration are detected and documented, which can be used to identify new best practices.

3 Related Work and Pattern Languages

According to Version One's 12th survey on the state of agile, companies are increasingly applying agile methods to large-scale projects [VersionOne 2018]. 52% of all respondents worked in organizations where more than half of the development teams worked with agile methods [VersionOne 2018]. Despite the relevance of this topic for practitioners, sound academic research is lacking, especially regarding to challenges and success factors [Dikert et al. 2016]. Some researchers recognized this gap and started to publish scientific papers, which are described below.

[Dikert et al. 2016] made a first attempt by conducting a systematic literature review of industrial large-scale agile transformations. They presented qualitative findings describing 35 reported challenges and 29 success factors from 42 different organizations. Challenge categories that received most attention were *agile difficult to implement*, *integrating non-development functions*, *change resistance*, and *requirements engineering challenges*. The most salient success factors were *management support*, *choosing and customizing the agile model*, *training and coaching*, and *mindset and alignment*. By means of a literature review, [Kalenda et al. 2018] identified practices, challenges, and success factors of large companies adopting agile methods. These findings were then compared and used to study a software company that was in the process of scaling agile methods. They identified four challenges, namely *resistance to change*, *quality assurance issues*, *integrating with non-agile parts of the organization*, and *too fast roll-out*. In addition, they determined four success factors: *unification of views and values*, *executive sponsorship and management support*, *company culture*, and *prior agile and lean experience*. In a previous study, we identified typical concerns of stakeholders and initiatives in large-scale agile development based on a structured literature review [Uludağ et al. 2018]. Based on the analysis of 73 papers, we identified 14 typical stakeholders in large-scale agile development, e.g., *development team*, *scrum master*, and *software architect*. In a subsequent step, we revealed typical concerns of respective stakeholders. In total, 79 challenges were identified and grouped into eleven challenge categories, which include *culture and mindset*, *enterprise architecture*, and *geographical distribution*, to name a few [Uludağ et al. 2018]. Our previous work constitutes the foundation of this paper since it also identified pattern candidates of some of our pattern language elements: *stakeholders*, *challenges*, and candidates for *methodology patterns*, *architecture principles*, *viewpoint patterns*, and *anti-patterns* [Uludağ et al. 2018]. Speaking of which, [Meszaros and Doble 1997] recommend reading other related pattern languages while writing patterns. By doing that, we identified some related pattern languages (see Table 1). Based on the comparison of related pattern languages, we identified the following shortcomings, which we aim to address with our proposed large-scale agile development pattern language:

- Only 10 out of 507 identified "potentially relevant" patterns focus on large-scale agile development.
- Our evaluation results in Section 6 show that practitioners ask for pattern languages that categorize patterns in the way they are executed. However, the analyzed pattern languages do not necessarily meet these expectations.
- The results of our evaluation in Section 6 show that a pattern language should include related stakeholders who apply patterns to address their concerns. Nevertheless, the concept of stakeholders is yet neglected, making the pattern language less practical for practitioners as they try to find relevant patterns for their specific roles as quickly and easily as possible.

4 Large-Scale Agile Development Pattern Overview

The application of agile methods on a large scale brings along unique challenges and difficulties [Boehm and Turner 2005; Dikert et al. 2016], e.g., increased number of stakeholders and, coordination complexity, and difficult architectural integration [Badampudi et al. 2013; Paasivaara and Lassenius 2014]. Tackling these is the key to reap the full benefits of agility in large-scale settings [Kettunen and Laanti 2008]. There are several scaling agile frameworks that pledge to resolve the aforementioned issues but are still in a nascent state [Alqudah and Razali 2016; Dingsøyr et al. 2019]. But even valuable research studies that provide explanations of how to address the challenges of large-scale agile development remain

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

Table 1: Overview of Related Pattern Languages

Source	Scope & goal	Focus on agile development	Number of patterns	Pattern categories	Pattern examples
[Coplien 1995]	Collection of patterns for shaping a new organization and its development processes	Partially	42	(1) Process patterns; (2) Organizational patterns	- Code Ownership - Gatekeeper - Fire Walls
[Harrison 1996]	Collection of patterns for creating effective software development teams	No	4	–	- Unity of Purpose - Diversity of Membership - Lock 'Em Up Together
[Beedle et al. 1999]	Collection of Scrum patterns	Yes	3	–	- Sprint - Backlog - Scrum Meetings
[Taylor 2000]	Collection of patterns for creating product software development environments	No	9	(1) Establishing a Production Potential; (2) Maintaining a Production Potential; (3) Preserving a Production Potential	- Deliverables to Go - Pulse - Bootstrapping
[Coplien and Harrison 2004]	Collection of organizational patterns that are combined into a collection of four pattern languages	Yes	94	(1) Project Management; (2) Piecemeal Growth; (3) Organizational Style; (4) People and Code	- Skill Mix - Demo Prep - Few Roles
[Elssamadisy 2008]	Collection of patterns for successfully adopting agile practices	Yes	38	(1) Feedback Practices; (2) Technical Practices; (3) Supporting Practices; (4) The Clusters	- Refactoring - Continuous Integration - Simple Design
[Beedle et al. 2010]	Collection of the most essential best practices of Scrum	Yes	11	–	- Daily Scrum - Sprint Backlog - Sprint Review
[Välimäki 2011]	Enhancing performance of project management work through improved global software project management practice	Partially	18	(1) Directing a Project; (2) Starting up a Project; (3) Initiating a Project; (4) Controlling a Stage; (5) Managing Stage Boundaries; (6) Closing a Project; (7) Managing Product Delivery; (8) Planning	- Collocated Kick-Off - Choose Roles in Sites - Iteration Planning
[Mitchell 2016]	Collection of patterns to address agile transformation problems	Yes	54	(1) Patterns of Method; (2) Patterns of Responsibility; (3) Patterns of Representation; (4) Anti-Patterns	- Limited WIP - Kanban Sandwich - Controlled Failure
[ScrumPLoP 2019]	Body of pattern literature around agile and Scrum communities	Yes	234 (10)	(1) Value Stream; (2) Team; (3) Sprint; (4) Process Improvement; (5) Product Organization; (6) Distributed Scrum; (7) Scaling Scrum; (8) Scrum Core; (9) Misc	- Scrum Master - Scrum of Scrums - Portfolio Standup

scarce [Bick et al. 2018]. Thus, following the idea of [Alexander 1977], the identification of recurring concerns and documentation of best practices in this context seems to be useful. Subsequently, we will introduce the structure of our pattern language (see Fig. 2). The pattern language distinguishes between three different types of patterns:

- **Coordination Patterns (C-Patterns)** define coordination mechanisms to address recurring coordination concerns, i.e., managing dependencies between activities, tasks or resources.
- **Methodology Patterns (M-Patterns)** define concrete steps to be taken to address given concerns.
- **Viewpoint Patterns (V-Patterns)** define proven ways to visualize information in form of documents, boards, metrics, models, and reports in order to address recurring concerns.

In addition, the pattern language includes four additional concepts:

- **Stakeholders** (in our context) are all persons who are actively involved in, have an interest in or are in some way affected by large-scale agile development [Uludağ et al. 2018].
- **Concerns** can manifest themselves in many forms, e.g., goals, responsibilities or risks [42010:2011(E) 2011].
- **Principles** are enduring and general guidelines that address given concerns by providing a common direction for action.
- **Anti-Patterns (A-Patterns)** describe typical mistakes and present revised solutions, which help pattern users to prevent these pitfalls.

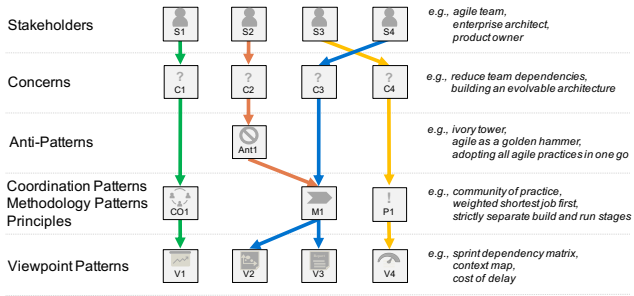

Figure 2: Conceptual overview of the proposed pattern language

Fig. 3 depicts the current version of our large-scale agile development pattern language, which can also be found on our prototypical web application[2]. The four highlighted nodes in Fig. 3 represent the four patterns that will be presented in Section 5. A detailed listing of the large-scale agile development patterns and concepts can be

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

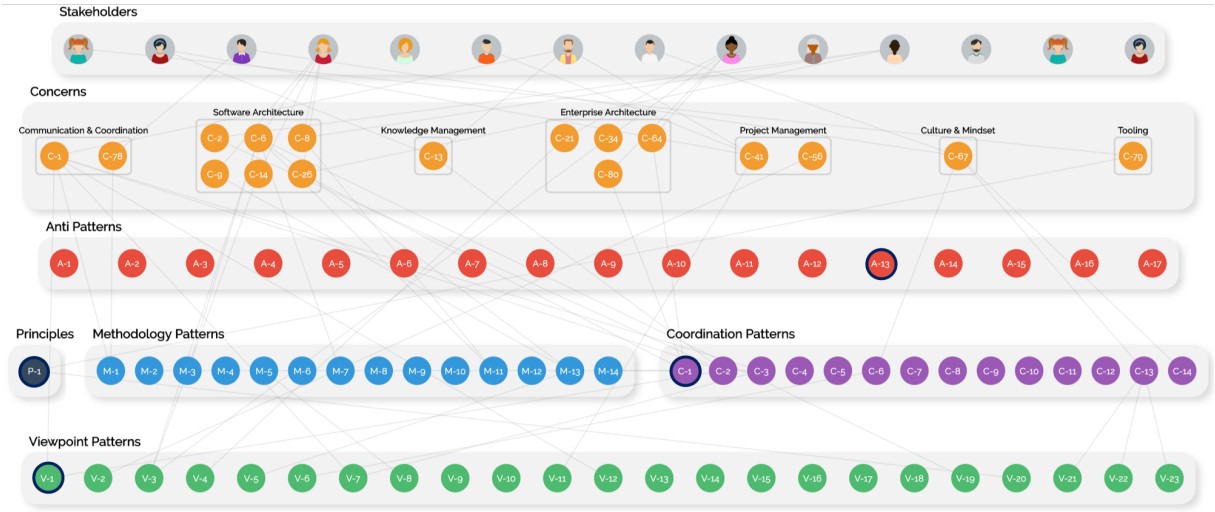

Figure 3: Current version of the large-scale agile development pattern language *

found in Appendix B.

Popular pattern forms include, among others, the Alexandrian Form, Gang of Four Form, Coplien Form, and Fowler Form [Ernst 2010; Fowler 2006]. All have specific benefits and limitations depending on the context [Ernst 2010]. Since there is no ideal pattern form, the author must consider his / her experience, the intention, and target audience when selecting either an existing form or creating a new one [Buschmann et al. 2007b; Ernst 2010]. According to [Fowler 2006], this choice is a personal decision and should also consider one's writing style and the ideas to be conveyed. Large-scale agile development patterns follow a template similar to [Buschmann et al. 1996; Ernst 2010]. Fig. 4 depicts the meta-model and the key

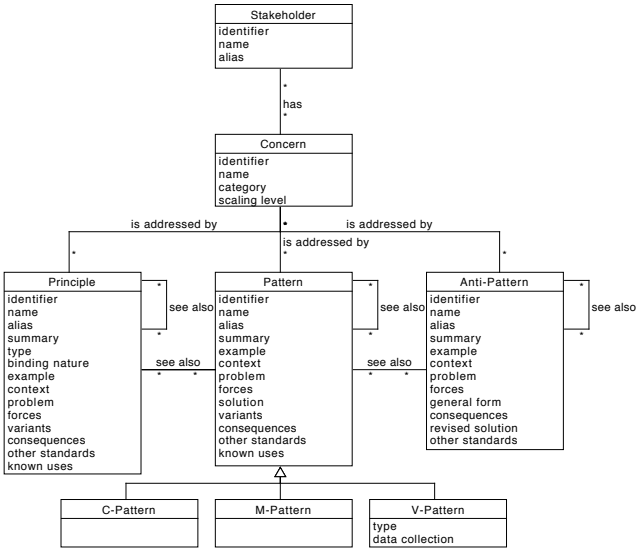

Figure 4: Meta-model of the proposed pattern language

elements used to document the concepts and patterns of the pattern language. All elements have an *identifier* and *name* which simplify referencing. A stakeholder has an additional section called *alias* that contains a list of synonyms and related role names. A concern has two additional sections called *category* and *scaling level* which denote the category and at which organizational level a concern occurs. Besides identifiers and names, principles, patterns, and anti-patterns consist of eight common sections: the *problem* and *context* sections describe problems and situations to or in which they apply. The *forces* section describes why the problem is difficult to solve. *summary* shortly recapitulates the solution. The *consequences* section contains associated benefits and liabilities, while the optional *other standards* and *see also* sections provide references to other solutions and frameworks. The *alias* section provides a list of synonyms. The *example* section illustrates the problem to be addressed. Principles and patterns consist of *variants* and *known uses* sections showing variants and alternatives as well as proven applications in practice. The *type* and *binding nature* sections are unique to principles and indicate their topic and whether they are recommended or mandatory. The *solution* section explains the recommended solution for a pattern. Specific to anti-patterns, the *general form* and *revised solution* sections include the recurring, not working solution and a revised solution presented. V-Patterns have *type* and *data collection* sections which show the visualization concept and collection processes required for their creation. Similar to [Buschmann et al. 2007a], we label our patterns with the star notation to denote our level of confidence in the pattern's maturity. Two stars mean that the pattern effectively addresses a genuine problem in its current form. One star denotes that the pattern addresses a real problem but needs to mature. No stars indicate that the pattern is a useful solution to an observed problem but requires significant revision. We will showcase in the following four patterns in order to highlight the differences between the presented pattern types and concepts, namely Strictly Separate Build and Run Stages (representing Principles), Community of Practice (showing C-Patterns), Iteration Dependency Matrix (demonstrating (V-Patterns), and Don't Use Agile as a Golden Hammer (illustrating A-Patterns).

5 Exemplifying the Large-Scale Agile Development Pattern Language

5.1 Principle: Strictly Separate Build and Run Stages (P-1) *

Principle Overview	
Alias	Build and Run
Summary	Strictly Separate Build and Run Stages ensures that an application's deployment phases are clearly separated.
Type	Software Architecture
Binding Nature	Recommended

5.1.1 Example

For one and a half years, four agile teams of RetailCo have been developing a cloud-based e-commerce platform. However, over the last few iterations, the operability and stability of the e-commerce platform of RetailCo have deteriorated dramatically. The agile teams of RetailCo have difficulties in delivering new versions of the platform in time. The platform also shows long downtimes due to massive traffic at bigger shopping-events such as Black Friday. Moreover, the product owner of the e-commerce platform receives customer complaints due to several bugs in the purchasing process.

5.1.2 Context

The release of stable and reliable cloud-native platforms in large-scale agile development is difficult because multiple agile teams work in parallel on the same software in complex setups aiming to release it as quickly and frequently as possible.

5.1.3 Problem

The following concern is addressed by Strictly Separate Build and Run Stages:

- *How can a cloud-native application be developed in a stable and timely manner?*

5.1.4 Forces

The following forces influence Strictly Separate Build and Run Stages:

- The code of a cloud-native application is changed at run-time, thus, is not reproducible.
- The build, run, and release phases of the deployment process of a cloud-native application are not self-contained, so the entire deployment workflow must be triggered.
- Deviations between the codes in the execution and staging environment are not traceable.

5.1.5 Variants

A possible variant for Strictly Separate Build and Run Stages would be to add a design step before the build stage. Within the design step, a high-level design of the upcoming small feature is created with every iteration. This can help to understand the dependencies of the application such as existing libraries the application is going to use.

5.1.6 Consequences

The following benefits of Strictly Separate Build and Run Stages are known:

- Traceability and reproducibility of releases
- Rollback to previous releases
- Faster releases of codebases to production
- Building a stable and reliable application
- No code is deployed without testing

The following liabilities of Strictly Separate Build and Run Stages are known:

- High degree of automation
- Deployment process is complicated

5.1.7 See Also

In order to measure the level of adherence of agile teams with Strictly Separate Build and Run Stages, the following V-Patterns should be considered:

- Deployment Time
- Deployment Frequency
- Change Frequency
- Mean Time to Change

5.1.8 Other Standards

Strictly Separate Build and Run Stages is also suggested by the Twelve-Factor App [Wiggins 2017].
Strictly Separate Build and Run Stages is extended by the Beyond the Twelve-Factor App [Hoffman 2016].

5.1.9 Known Uses

The following uses of Strictly Separate Build and Run Stages are known:

- GlobalInsureCo
- CarCo
- ITCo
- RetailCo
- PublicInsureCo

5.2 C-Pattern: Community of Practice (C-1) *

C-Pattern Overview	
Alias	Community, Guild
Summary	A Community of Practice are groups of people who share a concern, a set of problems, or a passion about a topic, and who deepen their knowledge and expertise in this area by interacting on an ongoing basis [Wenger et al. 2002].

5.2.1 Example

A vehicle dynamics development department of CarCo aims to transform its current traditional matrix organization to an agile organization by launching a large-scale endeavor with seven agile teams and more than 100 involved stakeholders. During this transformation process, CarCo has difficulties in aligning the agile teams working in the same department as they have no regular meetings on discussing common topics. Furthermore, the software architects of the large-scale agile endeavor recognize that the agile teams use some tools that are incompatible with each other making the integration of their sub products nearly impossible.

5.2.2 Context

Traditional agile approaches such as Scrum do not offer support large-scale cross-team coordination. Thus, establishing efficient coordination and knowledge sharing mechanisms between agile teams as well as between the experts in the teams might be difficult without having suitable knowledge sharing forums.

5.2.3 Problem

The following concern is addressed by Community of Practice:

- *How to create a platform for active knowledge sharing and discussion?*

5.2.4 Forces

The following forces influence Community of Practice:

- Facilitating shared context and knowledge across the organization is difficult
- Internal silos create gaps in knowledge and communication between agile teams

5.2.5 Solution

A Community of Practice meet regularly for knowledge sharing about a specific domain [Wenger et al. 2002]. The focus is to talk about practices that are applied and not to discuss theories. The participants of a Community of Practice are typically not from the same team, but from many different teams all across the organization. In the best case, many different practices can be presented and discussed, leading to a wide knowledge base. The participation in a Community of Practice is usually voluntary. In contrast to the M-Pattern Empowered Community of Practice, a traditional Community of Practice is not able to make binding decisions for the organization.

5.2.6 Variants

A Community of Practice can be set up for a variety of domains. A Community of Practice have been identified in the following domains: Architecture, Testing, Interfaces, Deployments, Leadership, and Infrastructure. In addition, a Community of Practice can also have some decision-making power for different topics, which is described in the M-Pattern Empowered Community of Practice.

5.2.7 Consequences

The following benefits of Community of Practice are known:

- Encouraging knowledge sharing for diverse topics
- Breaking up silos
- Enabling a culture of continuous improvement

The following liabilities of Community of Practice are known:

- Requiring an active involvement of participants
- Topics in the agenda could be too diverse and broad
- Providing right incentives to the participants is challenging

5.2.8 See Also

Community of Practice may be utilized in combination with the following M-Patterns:

- Consensus-Based Decision Making
- Empowered Community of Practice

5.2.9 *Other Standards*

Community of Practice is also recommended and practiced by the following scaling agile frameworks:

- Disciplined Agile Delivery [Ambler and Lines 2012]
- Large-Scale Scrum [Larman and Vodde 2016]
- Scaled Agile Framework [Scaled Agile 2019b]
- Spotify Model [Kniberg and Ivarsson 2012]

In addition, the Community of Practice is also described by [Coplien and Harrison 2004] as Community of Trust.

5.2.10 *Known Uses*

The following uses of Community of Practice are known:

- GlobalInsureCo
- CarCo
- ITCo
- RetailCo
- PublicInsureCo

5.3 V-Pattern: Iteration Dependency Matrix (V-1) *

V-Pattern Overview	
Alias	Sprint Dependency Matrix, Program Board
Summary	Iteration Dependency Matrix visualizes dependencies among teams for the upcoming iterations.
Type	Board

5.3.1 *Example*

The program manager and solution architect of RetailCo's e-commerce platform of RetailCo want to coordinate four agile teams for the five upcoming iterations. Thereby, they need a clear idea of which features or enablers will be done by which agile team in which iteration. In addition, they want to identify cross-team dependencies which might impact their delivery.

5.3.2 *Context*

In a Common Planning, cross-team dependencies between agile teams has to be detected and managed.

5.3.3 *Problem*

The following concerns are addressed by Iteration Dependency Matrix:

- *How to visualize dependencies between agile teams?*
- *How to coordinate multiple agile teams that work on the same product?*
- *How to consider integration issues and dependencies with other subsystems and teams?*

5.3.4 *Forces*

The following forces influence Iteration Dependency Matrix:

- Agile teams that work in the same large-scale agile development program have to be coordinated, i.e., delivering on the same cadence, timing the release of different features, and managing dependencies.
- Some dependencies between agile teams are not immediately visible.

5.3.5 *Solution*

Iteration Dependency Matrix visualizes dependencies between agile teams working on the same product for future iterations. The exemplary visualization in Fig. 5 shows team names as vertical headings, while iteration names are shown as horizontal headings. The blue rectangles represent features, while enablers are depicted as yellow rectangles. Important program milestones or events are depicted as orange rectangles. Each enabler or feature belongs to one team and one iteration. A feature may depend on other enablers. These dependencies are indicated by red strings.

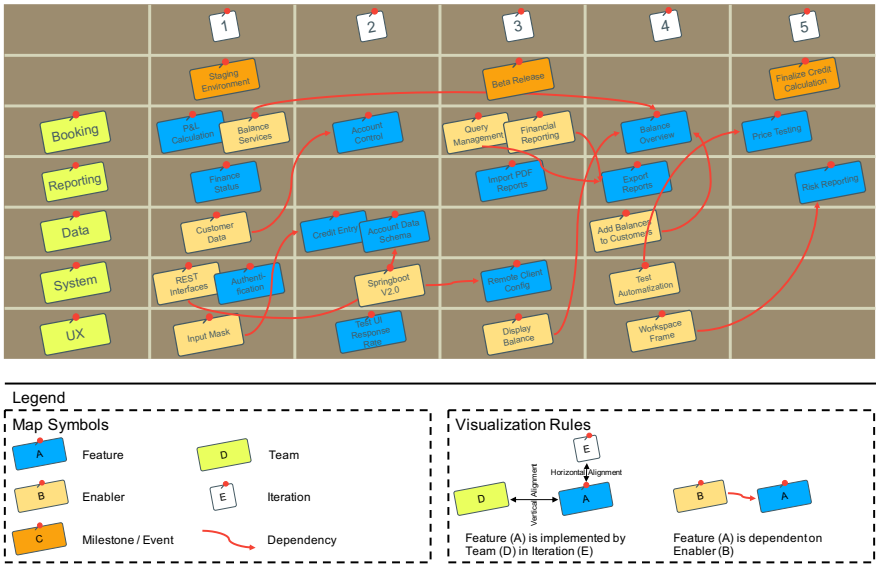

Figure 5: Exemplary view for Iteration Dependency Matrix

The underlying information model of Iteration Dependency Matrix is shown in Fig. 6.

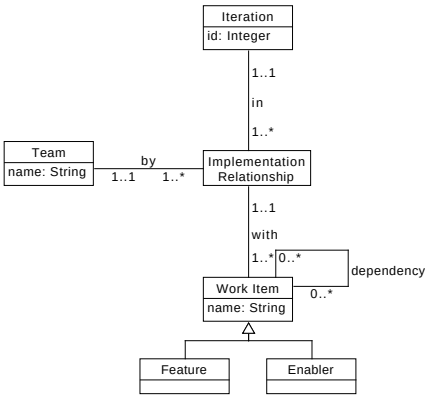

Figure 6: Underlying information model of Iteration Dependency Matrix

5.3.6 Variants

Additional variants exist for Iteration Dependency Matrix. First, enablers may be omitted if the organization does not make a differentiation between enablers and features, i.e., for budgetary reasons (see Fig. 7(b)). However, this variant is not advised as the importance of architectural improvements might be neglected. In addition, the Iteration Dependency Matrix could be simplified by omitting milestones or events that would happen in the upcoming iterations (see Fig. 7(b)). The Iteration Dependency Matrix can also be visualized digitally by the use of software development team collaboration tools. This variant has the advantage that useful information is stored in a digital format and that it can also be used in a distributed Common Planning.

5.3.7 Consequences

The following benefits of Iteration Dependency Matrix are known:

- Visualizing and tracking inter-team dependencies
- Revealing unplanned risks and dependencies
- Providing a visual overview of work to be done
- Optimizing development flow

The following liabilities of Iteration Dependency Matrix are known:

- High manual effort during its creation
- After the Common Planning, it might be abandoned
- Less effective when agile teams are remote

5.3.8 Data Collection

- *Frequency:* Features, enablers, and goals are usually written throughout the development process.
- *Responsible person:* Responsible persons are e.g., the business analyst, product manager, product owner, program manager or solution architect.
- *Data source:* Software development team collaboration tools.
- *Classes to be documented:* Team, enabler, feature, and iteration.

5.3.9 See Also

Iteration Dependency Matrix is used by the following C-Pattern:

- Common Planning

5.3.10 Other Standards

Iteration Dependency Matrix is also known as the Program Board in the Scaled Agile Framework [Scaled Agile 2019a].

5.3.11 Known Uses

The following uses of Iteration Dependency Matrix are known:

- CarCo
- RailCo
- BankingITCo
- RetailCo

- IndustryCo

(a) BankingITCo (b) RetailCo

Figure 7: Two observed Iteration Dependency Matrices

5.4 A-Pattern: Don't Use Agile as a Golden Hammer (A-13) *

A-Pattern Overview	
Alias	Law of the Instrument, Law of the Hammer, Maslow's Hammer, One Size Fits All
Summary	Don't Use Agile as a Golden Hammer shows why it is not advisable to use agile methods in order to solve many kinds of problems.

5.4.1 Example

Success stories of the autonomous driving department with the application of agile methods reached the IT department of LuxCarsCo. The IT management decided to transform all current IT projects to agile.

5.4.2 Context

Agile methods have become very popular in the last few years. Deciding when agile methods are appropriate for a particular project is a difficult task as different aspects have to be considered, e.g., project size, project objectives and requirements, project team, technological realization, and so on.

5.4.3 Problem

The following concerns are addressed by Don't Use Agile as a Golden Hammer:

- *How to choose the correct software development approach?*
- *How to decide whether agile methods should be used for a given project?*

5.4.4 Forces

The following forces occur in the context of Don't Use as a Golden Hammer:

- Several successes tempt the organization to solely use agile methods
- Large investment has been made in agile training
- Lack of motivation to explore alternative methods

5.4.5 General Form

A software development team has gained a lot of attention within the organization due to its success by using agile methods. As a result, the organization believes that every new product should be developed by the use of agile methods. In some cases, the use of agile methods will not solve the problem.

5.4.6 Variants

A possible variant for Don't Use Agile as a Golden Hammer is that not only the management of an organization forces the use of agile methods for software projects, but also existing software development teams obsessively operate use of agile methods.

5.4.7 Consequences

The following benefits of Don't Use Agile as a Golden Hammer are known:

- Management-Buy-In

The following liabilities of Don't Use Agile as a Golden Hammer are known:

- Failed projects

5.4.8 Revised Solution

The most important aspect of the revised solution is: Try to avoid Don't Use Agile as a Golden Hammer. This can be done by using evidence why the use of agile methods might not be appropriate, i.e., in simple projects in which the technological realization and requirements are known (see Fig. 8). Don't Use Agile as a Golden Hammercan also be avoided when a conscious effort towards the exploration of alternative software development methods is made. Use Community of Practice to facilitate the exchange of ideas and experiences and to understand their rationale for applying agile methods.

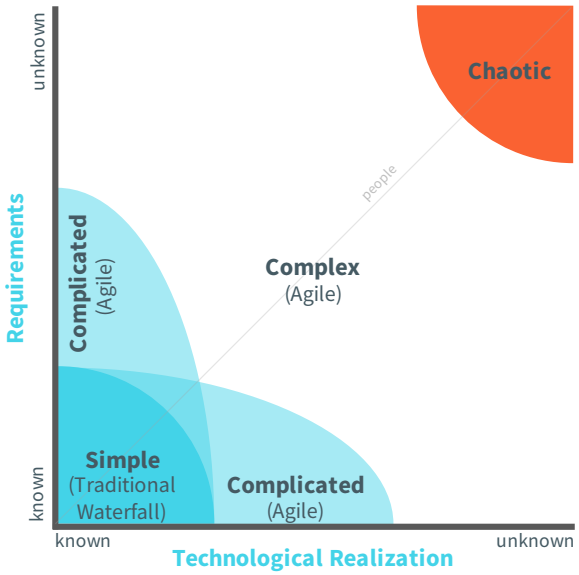

Figure 8: Stacey Matrix [Stacey 1996] adapted to software development

6 Evaluation

We interviewed 14 large-scale agile development experts from 10 organizations to assess the structure and practical relevance of the proposed pattern language. The companies are mainly headquartered in Germany, predominantly active in the IT, insurance, and retail sector, and employ around 2,000 to 200,000 people. Most of them started with large-scale agile development a few years ago. We prepared a questionnaire (see Appendix A) containing scale response (on a five-point Likert scale) and open-ended questions to collect expert feedback on our proposed pattern language. Table 2 lists the roles and professional experiences of all interviewed experts with large-scale agile development. During the interviews,

Table 2: Interview partners for evaluating the proposed pattern language

No	Alias	Main role	Professional experience (in years)
1	EA1	Enterprise Architect	1-3
2	PM1	Project Manager	1-3
3	AD1	Agile Developer	3-6
4	AC1	Agile Coach	3-6
5	EA2	Enterprise Architect	3-6
6	EA3	Enterprise Architect	3-6
7	SA1	Solution Architect	3-6
8	PA1	Platform Architect	>6
9	PKE1	Principal Key Expert	3-6
10	AC2	Agile Coach	3-6
11	AC3	Agile Coach	>6
12	PO1	Product Owner	>6
13	EA4	Enterprise Architect	3-6
14	AD2	Agile Developer	<1

we provided a definition of each concept and presented the overall structure of the pattern language by demonstrating five examples (including their relationships to recurring concerns and other concepts) using our prototypical web application. In the following, we present the results of our evaluation[3] (see Fig. 9).

Overall evaluation of the pattern language concepts: Stakeholders were rated as the most valuable concept of the pattern language ($\mu = 1.00$; $\sigma = 0.0$). Concerns were rated the second most valuable concept ($\mu = 1.14$; $\sigma = 0.18$). The inclusion of V-Patterns was considered very valuable ($\mu = 1.43$; $\sigma = 0.41$). The usefulness of M-Patterns was received high ($\mu = 1.50$; $\sigma = 0.37$). The concept of C-Patterns was considered very useful ($\mu = 1.71$; $\sigma = 0.44$). Interviewees broadly agreed that principles should be included in the pattern language ($\mu = 1.79$; $\sigma = 0.60$). A-patterns received a positive feedback ($\mu = 2.00$; $\sigma = 0.42$). Initiatives were rated less useful ($\mu = 2.50$; $\sigma = 0.65$)

Stakeholders: The concept of stakeholders was rated indispensable (AD1), as it provides transparency about relevant roles and their concerns (EA1, EA3, PM1). One interviewee also stated that: *"patterns are developed for stakeholders"* (EA2).

Concerns: Main arguments for including concerns in the pattern language were that they provide a good starting point for decision making (EA1) and that users can directly access relevant patterns by navigating stakeholder-specific concerns (EA1, PM1). AC1 also stated that *"concerns are one of the best ways for describing stakeholder roles"*. Other respondents expressed concerns as the central motivation of the pattern language (AC1, AD2, EA2, EA3, SA1). Downsides were that concerns are too detailed (AC2), and should be made less specific (EA4). AC3 added that some concern categories such as coaching, training, human resources, etc. were missing.

V-Patterns: The interviewees mentioned the following benefits of including V-Patterns in the pattern language: they can be used as a communication medium (EA1), create a common ground between stakeholders (PO1), help with decision-making (EA4), and provide early feedback and transparency (PA1). EA2 considers V-Patterns in large-scale agile development to be as important as V-Patterns in the EAM pattern catalog, which he uses regularly [Ernst 2010; Khosroshahi et al. 2015]. AD1 and AC2 added that V-Patterns are useful as a reference book since they provide a wide-ranging toolkit for solving recurring concerns. EA3 recommended that V-Patterns should be extended by an additional section to document underlying data collection processes required for their creation.

M-Patterns: Main arguments for including M-Patterns in the pattern language were that they provide concrete guidance on how to solve problems (AC1, AD2 EA3, SA1) and that they *"represent the core of the pattern language"* (SA1). According to PO1, M-Patterns can be used as change vehicles within the organization and depend on the organizational maturity level to be applied. AD1 and EA2 claimed that the application could be difficult and may require some adaptions to be used. EA4 proposed to merge the concepts of C- and M-Patterns as *"they are very similar"*. In contrast, 71.43% of the interviewees regarded the separation of C- and M-Patterns as helpful.

C-Patterns: Interviewees stated that C-Patterns provide best practices and proven mechanisms to common coordination and communication concerns (AD2, EA2, PA1, PM1). They also show which persons have to meet and enable subject-related alignment and coordination (EA1). PKE1 added that the *variants* section of C-Patterns could be helpful to document alternative meetings formats. Although AD1 and EA3 perceived C-Patterns as very relevant in practice, they claimed that the content discussed in meetings are very complex and highly context specific, thus, making them difficult to generalize.

Principles: Positive arguments for including principles in the pattern language were that they reduce the complexity of large-scale agile development endeavors (AD2) and provide a common direction and guidance for agile teams (AD2, EA1, EA3, PO1). Interviewees highlighted that principles should be stated explicitly (AC2, PA1) as they are helpful for new employees and improve understandability (AC2). PO1 added that principles foster mindset control without prescribing concrete methodologies, thus stimulating understanding and empowerment of agile teams (P01). In this context, EA4 perceived architecture principles as essential for the success of large-scale agile development endeavors. The relationship between principles and V-Patterns was considered valuable (AC2, EA1) as the compliance of agile teams with principles can be ensured by

[3] In this paper, we do not present the evaluation results of the relationships between the pattern concepts, because their usefulness was mainly influenced by the importance the interviewees associated with the respective pattern concepts.

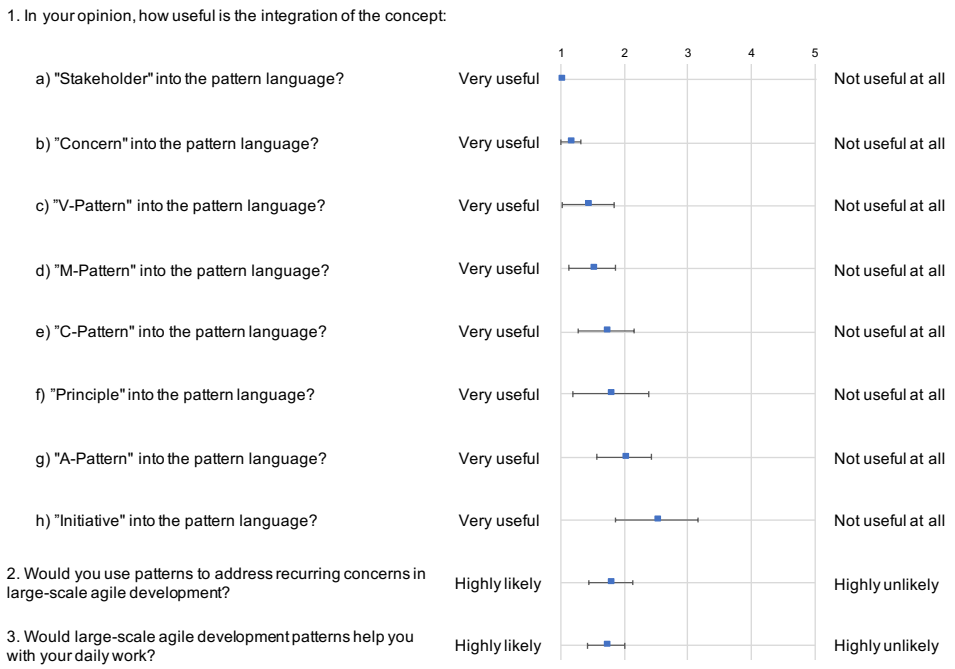

Figure 9: Evaluation results of the large-scale agile development pattern language

the use of KPIs (EA1). Although other participants acknowledged the usefulness, they rated them less applicable in this context (AD1, EA2, PM1, SA1). AC1 and EA3 had two recommendations. First, the binding nature should be included indicating mandatory or recommended principles. Second, principles should not have a *solution* section as this is the task of an M-Pattern.

A-Patterns: A-Patterns prevent pattern users from running into the same traps (EA1), show what not to do (PM1), and raise awareness of failed approaches (EA2). AD2 added that A-Patterns save time because *"you don't have to relive the same problem"*. PO1 perceived them as important but noted that *"people still make these mistakes"*. EA4 mentioned the potential disadvantage that *"everyone must understand how it is meant"*. Some interviewees questioned the usefulness of A-Patterns as they are not connected with other concepts of the pattern language (AD1, EA3, SA1). We asked whether A-Patterns should be connected with other concepts and 92.68% of interviewees replied positive. They agreed that A-Patterns should have a *problem* section, thus, being connected with concerns and stakeholders, to provide A-Patterns with more context. This enables more usability, as users can find stakeholder-relevant A-Patterns faster in comparison to other pattern languages without this connection. They also stated that the *revised solution* section should refer to C-Patterns, M-Patterns, and V-Patterns, to find appropriate solutions.

Initiatives: Some respondents said initiatives are helpful when concerns cannot be directly related to stakeholders (AC1, AD1) and if concerns occur on different organizational levels (EA1, PKE1). On the other hand, they were considered redundant (PM1) since they do not provide additional information or value (AC1, PM1, SA1). Two respondents found this concept too abstract (AC1) or not relevant for industry (EA2). In addition, they were also confused about the denotation of initiatives (AC2, AC3, PA1). Several interviewees recommended removing initiatives from the pattern language structure as it increases complexity (EA2, PKE1, PO1, SA1).

Usefulness and Support of Patterns: The majority of respondents indicated that they would use patterns to address recurring concerns in large-scale agile development.

Based on the evaluation, we made the following adjustments to the large-scale agile development pattern language structure:

- Initiatives are removed and are instead represented by the *scaling level* attribute of concerns.
- Principles are extended by a *binding nature* attribute to indicate recommended or mandatory principles.
- The *solution* section of principles is removed so that they remain generic and refer to related M-Patterns.
- A-Patterns are connected with concerns, stakeholders, C-, M-, and V-Patterns.
- V-Patterns are extended by an optional data collection attribute so that recommended data collection processes are described.

As already indicated in Section 4, the mentioned adjustments are already incorporated in the overview and meta-model of the pattern language shown in Fig. 2 and Fig. 4.

7 Conclusion and Outlook

Given the heterogeneity of large-scale projects, the nascent state of scaling agile frameworks and empirical studies [Dikert et al. 2016; Dingsøyr et al. 2019], large-scale agile development is a field susceptible to practice-driven design research. The proposed language provides the structure for documenting practice-proven solutions

to recurring large-scale agile development concerns. The pattern language includes typical stakeholders, their concerns, principles, M-Patterns, V-Patterns, C-Patterns, and A-Patterns. It also links to other standards to allow easy integration and comparison. This pattern language was evaluated by interviewing 14 large-scale agile experts from 10 organizations and by observing and documenting new patterns, four of which were presented in this paper.

The results of this paper provide compelling directions for future research. We will continue to collect data from our large-scale agile development workshops [Uludağ 2019] and case studies with industry partners. In parallel, we will conduct structured interviews with different types of stakeholders, such as agile coaches, enterprise architects or product owners, to identify role-specific concerns and pattern candidates. Based on a structured survey among companies worldwide, we will publish the *Large-Scale Agile Development Pattern Catalog* containing concerns and patterns. In future work, we will assist our industry partners to select relevant patterns and to introduce them in their organizations. This will help us to evaluate the pattern implementations in practice and to observe changing pattern implementations. With this, we also aim to close the research activity cycle of the PDR method [Buckl et al. 2013].

Acknowledgements

This work has been sponsored by the *German Federal Ministry of Education and Research (BMBF)* via the Software Campus Project SaM-IT 01IS17049 project.

References

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

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

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

Scott Ambler and Mark Lines. 2012. *Disciplined agile delivery: A practitioner's guide to agile software delivery in the enterprise.* IBM Press.

Deepika Badampudi, Samuel A. Fricker, and Ana M. Moreno. 2013. Perspectives on Productivity and Delays in Large-Scale Agile Projects. In *Agile Processes in Software Engineering and Extreme Programming*, Hubert Baumeister and Barbara Weber (Eds.). Springer, Berlin, 180–194.

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

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

Saskia Bick, Kai Spohrer, Rashina Hoda, Alexander Scheerer, and Armin Heinzl. 2018. Coordination Challenges in Large-Scale Software Development: A Case Study of Planning Misalignment in Hybrid Settings. *IEEE Transactions on Software Engineering* 44, 10 (2018), 932–950.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Kevin Hoffman. 2016. *Beyond The Twelve-Factor App.* O'Reilly Media, Sebastopol.

Emam Hossain, Muhammad A. Babar, and Hye-Young Paik. 2009. Using Scrum in Global Software Development: A Systematic Literature Review. In *4th International Conference on Global Software Engineering.* IEEE, Limerick, 175–184.

Irum Inayat, Siti S. Salim, Sabrina Marczak, Maya Daneva, and Shahaboddin Shamshirband. 2015. A systematic literature review on agile requirements engineering practices and challenges. *Computers in human behavior* 51 (2015), 915–929.

Martin Kalenda, Petr Hyna, and Bruno Rossi. 2018. Scaling agile in large organizations: Practices, challenges, and success factors. *Journal of Software: Evolution and Process* 30, 10 (2018), e1954. https://doi.org/10.1002/smr.1954

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

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

Pouya A. Khosroshahi, Matheus Hauder, Alexander W. Schneider, and Florian Matthes. 2015. *Enterprise Architecture Management Pattern Catalog Version 2.0.* Technical Report. Chair of Software Engineering for Business Information Systems (sebis), Technical University of Munich.

Henrik Kniberg and Anders Ivarsson. 2012. Scaling Agile @ Spotify.

Philippe Kruchten. 2013. Contextualizing Agile Software Development. *Journal of Software: Evolution and Process* 25, 4 (2013), 351–361.

Craig Larman and Bas Vodde. 2016. *Large-Scale Scrum: More with LeSS.* Addison-Wesley Professional.

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

Gerard Meszaros and Jim Doble. 1997. A Pattern Language for Pattern Writing. In *Pattern Languages of Program Design 3*, Robert C. Martin, Dirk Riehle, and Frank Buschmann (Eds.). Addison-Wesley, Boston, 529–574.

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

Sridhar Nerur, RadhaKanta Mahapatra, and George Mangalaraj. 2005. Challenges of Migrating to Agile Methodologies. *Commun. ACM* 48, 5 (2005), 72–78.

Robert L. Nord, Ipek Ozkaya, and Philippe Kruchten. 2014. Agile in Distress: Architecture to the Rescue. In *15th International Conference on Agile Software Development.* Springer, Berlin, 43–57.

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

Knut H. Rolland, Brian Fitzgerald, Torgeir Dingsøyr, and Klaas-Jan Stol. 2016. Problematizing Agile in the Large: Alternative Assumptions for Large-Scale Agile Development. In *37th International Conference on Information Systems.* Association for Information Systems, Dublin.

Scaled Agile. 2019a. PI Planning. https://www.scaledagileframework.com/pi-planning. Accessed: 2019-02-02.

Scaled Agile. 2019b. Scaled Agile Framework. https://www.scaledagileframework.com. Accessed: 2019-02-02.

Alexander W. Schneider and Florian Matthes. 2015. Evolving the EAM Pattern Language. In *20th European Conference on Pattern Languages of Programs.* ACM, New York, 45:1–45:11.

ScrumPLoP. 2019. Published Patterns. https://sites.google.com/a/scrumplop.org/published-patterns/. Accessed: 2019-02-02.

Ralph Stacey. 1996. *Strategic Management & Organizational Dynamics: The Challenge of Complexity.* Pitman Publishing, London.
Paul Taylor. 2000. Capable, productive, and satisfied: Some organizational patterns for protecting productive people. In *Pattern Languages of Program Design 4*, John M. Vlissides, James O. Coplien, and Norman L. Kerth (Eds.). Addison-Wesley, Boston, 611–636.
Ömer Uludağ. 2019. Scaling Agile Practices Workshops. https://wwwmatthes.in.tum.de/pages/1lihu1sjq8jpk/Scaling-Agile-Practices-Workshops. Accessed: 2019-02-02.
Ömer Uludağ, Martin Kleehaus, Christoph Caprano, and Florian Matthes. 2018. Identifying and Structuring Challenges in Large-Scale Agile Development Based on a Structured Literature Review. In *22nd International Enterprise Distributed Object Computing Conference*. IEEE, Stockholm, 191–197.
Antti Välimäki. 2011. *Pattern Language for Project Management in Global Software Development.* Tampere University of Technology, Tampere.
VersionOne. 2018. *12th Annual State of Agile Report.* Technical Report. VersionOne.
Etienne Wenger, Richard Arnold McDermott, and William Snyder. 2002. *Cultivating communities of practice: A guide to managing knowledge.* Harvard Business Press.
Adam Wiggins. 2017. The Twelve-Factor App. https://12factor.net/. Accessed: 2019-02-20.

A Questionnaire

A.1 General questions

Name, Organization, Role description, Personal large-scale agile development experience level, Organizational large-scale agile development experience level, Operation level, Country of headquarters, Sector, Number employees

A.2 Pattern language concepts

In your opinion, how useful is the integration of the concepts: "Stakeholder", "Initiative", "Concern", "Principle", "Coordination Pattern", "Methodology Pattern", "Viewpoint Pattern", and "Anti-Pattern" into the pattern language? (five-point Likert scale question)

Why is the concept: "Stakeholder", "Initiative", "Concern", "Principle", "Coordination Pattern", "Methodology Pattern", "Viewpoint Pattern", and "Anti-Pattern" useful / not useful for you? (open-ended question)

Does the differentiation between "Coordination Pattern" and "Methodology Pattern" help you? (yes-no question + open-ended question)

In your opinion, should the "Anti-Pattern" concept be connected to another concept from the pattern language? (yes-no question) If yes, with which concept from the pattern language should the concept "Anti-Pattern" be connected? (open-ended question)

A.3 Relationship between pattern language concepts

In your opinion, how useful is the relationship between the concepts: "Stakeholder" and "Concern", "Initiative" and "Concern", "Concern" and "Principle", "Concern" and "Coordination Pattern", "Concern" and "Methodology Pattern", "Concern" and "Viewpoint Pattern", "Principle" and "Viewpoint Pattern", "Coordination Pattern" and "Viewpoint Pattern", "Methodology Pattern" and "Viewpoint Pattern", and "Coordination Pattern" and "Methodology Pattern"? (five-point Likert scale question + open-ended question)

A.4 Discussion

Is there another concept you would like to see in the pattern language? (open-ended question)

Would you use patterns to address recurring concerns in large-scale agile development? (five-point Likert scale question)

Would large-scale agile development patterns help you with your daily work? (five-point Likert scale question)

B Current patterns and concepts in the pattern language *

B.1 Stakeholders

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

B.2 Concerns

- **C-1:** *How to coordinate multiple agile teams that work on the same product?*
- **C-2:** *How to consider integration issues and dependencies with other subsystems and teams?*
- **C-6:** *How to manage technical debts?*
- **C-8:** *How to ensure that non-functional requirements are considered by the development team?*
- **C-9:** *How to find the right balance between architectural improvements and business value?*
- **C-13:** *How to share common vision?*
- **C-14:** *How to create a proper upfront architecture design of the system?*
- **C-21:** *How to manage dependencies to other existing environments?*
- **C-26:** *How to align and communicate architectural decisions?*
- **C-34:** *How to ensure the reuse of enterprise assets?*
- **C-41:** *How to deal with unplanned requirements and risks?*
- **C-56:** *How to define clear roles and responsibilities?*
- **C-64:** *How to define a lightweight formal review process for new technologies?*
- **C-67:** *How to encourage development teams to talk about tasks and impediments?*
- **C-78:** *How to synchronize sprints in the large-scale agile development program?*
- **C-79:** *How to ensure that the development phases are clearly separated and executed in an iterative fashion?*
- **C-80:** *How to ensure that development teams comply with architecture principles?*

B.3 A-Patterns

- **A-1:** Don't Force Traditional Project Management Concepts to Agile Software Development
- **A-2:** Don't Adopt All Agile Practices in One Go
- **A-3:** Don't Misinterpret the Meaning of Working Software Over Comprehensive Documentation
- **A-4:** Don't Spare Expenses on Agile Mindset Education
- **A-5:** Don't Assume a Tacit, Implicit Understanding of Architecture, its Scope, and the Architect's Role and Responsibility
- **A-6:** Don't Build an Ivory Tower
- **A-7:** Don't Overshoot Coordination Meetings
- **A-8:** Don't Force Traditional Job Behaviors to Agile Software Development
- **A-9:** Don't Put Individual Goals Over Team Goals
- **A-10:** Don't Creep Old Bureaucracy in Agile Software Development
- **A-11:** Don't Mix Old Approaches with Agile Software Development Approaches
- **A-12:** Don't Use Agile Practices out-of-the-box without Adapting to your Own Needs
- **A-13:** Don't Use Agile as a Golden Hammer
- **A-14:** Don't Try to Reduce the Amount of Communication in Large-Scale Agile Development Programs by Documentation
- **A-15:** Don't Consider Knowledge Sharing Strategies and Projects in Isolation
- **A-16:** Don't Add New Developers into New Teams, Instead Add Them into Existent Ones
- **A-17:** Don't Develop a Single Requirement Involved Multiple Agile Teams in Different Locations

B.4 Principles

- **P-1:** Strictly Separate Build and Run Stages

B.5 M-Patterns

- **M-1:** Cadence-Based Development
- **M-2:** Collaborative Establishment of Architecture Principles
- **M-3:** Continuous Delivery Pipeline
- **M-4:** DevOps
- **M-5:** Kanban
- **M-6:** Scrum
- **M-7:** Architectural Runway
- **M-8:** Extreme Programming
- **M-9:** ScrumXP
- **M-10:** ScrumBan
- **M-11:** Capturing NFRs in Definition of Done
- **M-12:** Architecture Spike
- **M-13:** Weighted Shortest Job First Prioritization
- **M-14:** Collaborative Adoption of new Technologies

B.6 C-Patterns

- **C-1:** Community of Practice
- **C-2:** Common Planning
- **C-3:** Scrum-of-Scrums
- **C-4:** Center of Excellence
- **C-5:** Sprint Planning
- **C-6:** Common Retrospective
- **C-7:** Sprint Review
- **C-8:** Backlog Refinement
- **C-9:** System Demo
- **C-10:** Inspect and Adapt
- **C-11:** Sprint
- **C-12:** Daily Standup
- **C-13:** Sprint Retrospective
- **C-14:** Supervision

B.7 V-Patterns

- **V-1:** Iteration Dependency Matrix
- **V-2:** Responsibility Assignment Matrix
- **V-3:** Architecture Solution Space
- **V-4:** Business Capability Map
- **V-5:** Weightest Shortest Job First
- **V-6:** Global Impediment Board
- **V-7:** SoS Board
- **V-8:** Kanban Board
- **V-9:** Portfolio Canvas
- **V-10:** SWOT Analysis
- **V-11:** ROAM Board
- **V-12:** Objectives Board
- **V-13:** Business Object Model
- **V-14:** Application Interface Map
- **V-15:** Solution Context Map
- **V-16:** Value Stream Map
- **V-17:** Persona
- **V-18:** Burndown Chart
- **V-19:** Technical Debt Backlog
- **V-20:** Speed to Market
- **V-21:** Impediment Board
- **V-22:** Starfish
- **V-23:** Run the Sail Boat