Received: 13 September 2017 | Revised: 14 March 2018 | Accepted: 15 March 2018
DOI: 10.1002/smr.1954

WILEY Journal of Software: Evolution and Process

PRACTICE PAPER

Scaling agile in large organizations: Practices, challenges, and success factors

Martin Kalenda[1] | Petr Hyna[2] | Bruno Rossi[1]

[1]Faculty of Informatics, Masaryk University, Brno, Czech Republic
[2]Kentico, Brno, Czech Republic

Correspondence
Bruno Rossi, Faculty of Informatics, Masaryk University, Brno, Czech Republic.
Email: brossi@mail.muni.cz

Abstract

Context: Agile software development has nowadays reached wide adoption. However, moving agile to large-scale contexts is a complex task with many challenges involved. *Objective:* In this paper, we review practices, challenges, and success factors for scaling agile both from literature and within a large software company, identifying the most critical factors. *Method:* We conduct a focused literature review to map the importance of scaling practices, challenges, and success factors. The outcome of this focused literature review is used to guide action research within a software company with a view to scaling agile processes. *Results:* Company culture, prior agile and lean experience, management support, and value unification were found to be key success factors during the action research process. Resistance to change, an overly aggressive roll-out time frame, quality assurance concerns, and integration into preexisting nonagile business processes were found to be the critical challenges in the scaling process. *Conclusion:* The action research process allowed to cross-fertilize ideas from literature to the company's context. Scaling agile within an organization does not need to follow a specific scheme, rather the process can be tailored to the needs while keeping the core values and principles of agile methodologies.

KEYWORDS

action research, agile adoption, large-scale agile, Large-Scale Scrum (LeSS), Scaled Agile Framework (SAFe)

1 | INTRODUCTION

Agile software development is very popular nowadays, as its application in many different contexts shows.[1,2] However, traditional agile methods were designed with a single team in mind and were not meant to face scalability challenges.[3] In this sense, "scaling" can be described as "a continuous process of transferring, translating, and transforming knowledge across various communities and individuals" (Rolland[4] citing Carlile[5,6])—placing large emphasis on the communication needs during a scaling process.

Scaling agile software development in large organizations is complex and poses several challenges.[7-11] Large projects require appropriate coordination and communication between teams,[12-14] dependencies between teams need to be managed, other nonagile units need to be involved,[15] and the right people need to be part of the process.[16,17] Recent research reports that most of the goals and practices for scaling agile are domain independent, listing as key factors *challenges in coordinating multiple teams*, *difficulties with managing requirements*, *problems in adaptation with the organizational structure*, and *issues in understanding agile concepts along the value chain*.[1] Additionally, *customer involvement*, *software architectural concerns*, and *interteam coordination* were also reported,[18] together with *challenges in coordinating the work of several teams*.[19]

In general, *training personnel*, *informing and engaging people in the process*, and *involving actors that can push the process further* were general success factors found in case studies related to agile scalability.[9,20] Adopting new scaling practices is also challenging, as there are many cross-cutting factors that can impact and have been found in technology adoption studies.[21-26] In particular, *resistance to change* and *weak management support* can play a relevant role.[22,25,26] As it has been observed, "real agile development with Scrum implies a deep change to become an agile organization; it is not

Abbreviations: LeSS, Large Scale Scrum; SAFe, Scaled Agile Framework; CoP, Communities of Practice; SoS, Scrum of Scrums; CTO, Chief Technology Officer; CEO, Chief Executive Officer; UX, User eXperience; QA, Quality Assurance; CMS, Content Management System;

a practice, it is an organizational design framework. Start a large-scale agile Scrum adoption by ensuring leadership understands the organizational implications, and they have been proven adoptable in the small scale."[27,28]

Scaling agile and finding the best scaling agile practices are among the most relevant research topics in terms of interest for practitioners.[9,29,30] We are, however, far away from reaching consensus on the best set of practices given a context.[4,9] We know that scaling agile is particularly challenging in the distributed development context,[10,31] for the development of safety critical systems in regulated environments,[2] for continuous value delivery,[32] and due to contradictions that might emerge in large bureaucratic organizations.[33]

The goal of this paper is to collect more evidence about impact factors during an agile scaling process, as we need more evidence about practices that work best in a given context,[1,34] collected as the process happens and not ex-post,[4] possibly to research variations over time.[18] We approach the goals by firstly evaluating existing methods and frameworks for scaling agile development. We select 2 of them—Scaled Agile Framework (SAFe)[35,36] and Large-Scale Scrum (LeSS)[27]—that we examine more deeply. Using the knowledge assessed from this evaluation, we extract common practices that these methods and frameworks use to scale agile development. Secondly, we conduct a research of studies that examine experiences of large companies adopting agile methods. In this research, we determine challenges and success factors of these adoptions, as well as study scaling practices that these companies used. Finally, we conduct an action research study[37,38] in a large company scaling-up the agile software development processes, to derive challenges, success factors, and specific scaling practices.

In this paper, we aim at answering 6 main research questions. The first 3 are derived from literature dealing with large-scale projects:

RQ1. *Which agile scaling **practices** reported in literature did large companies (embracing practices from SAFe, LeSS) use?*
RQ2. *Which **challenges** reported in literature did large companies (embracing practices from SAFe, LeSS) face when adopting scaled agile development?*
RQ3. *Which **success factors** reported in literature helped large companies (embracing practices from SAFe, LeSS) to adopt and scale agile development?*

At the company-specific level, we aimed at answering the following:

RQ4. *Which agile scaling **practices** were the most effective in the company's context?*
RQ5. *Which **challenges** did the company face when adopting a scaled agile development process?*
RQ6. *Which **success factors** were relevant in the company to adopt a scaled agile development process?*

The paper is structured as follows. Section 2 proposes the research methodology, based on a focused literature review (step 1) used to support an action research study (step 2) conducted within large software company. Section 3 presents the background about scaling practices, discussing 6 frameworks for scaling agile, extracting common practices used in other parts of the research. Section 4 presents the results from the main literature review (step 1) to collect evidence about practices, challenges, and success factors from previous research (RQ1, RQ2, and RQ3). These factors are used during the action research process (step 2), presented in Section 5, discussing the context, the needs to scale agile development processes, and the findings in terms of practices, success factors, and challenges within the company (RQ4, RQ5, and RQ6). Section 6 presents the summary of the findings with a comparison with other papers reviewing primary studies on scaling agile processes. Section 7 provides the conclusions.

2 | RESEARCH METHODOLOGY

We followed a research process based on 2 main steps. The first step (step 1, Figure 1) was a scoped literature review. The goal of the literature review was to make aware industrial study partners of impact factors in scaling agile studies. We were aware of the excellent and exhaustive review by Dikert et al,[34] and our goal was not to provide another extensive review in the agile context (eg, Schön et al[39] and Torrecilla-Salinas et al[40]). Our aim was to disseminate the company with material that could be easy to understand and consult, providing a good starting point for the second step.

The literature review was useful to define practices, challenges, and success factors that were found in previous research. After the initial identification of common practices from SAFe and LeSS frameworks, we run the literature review to identify most used practices, challenges faced,

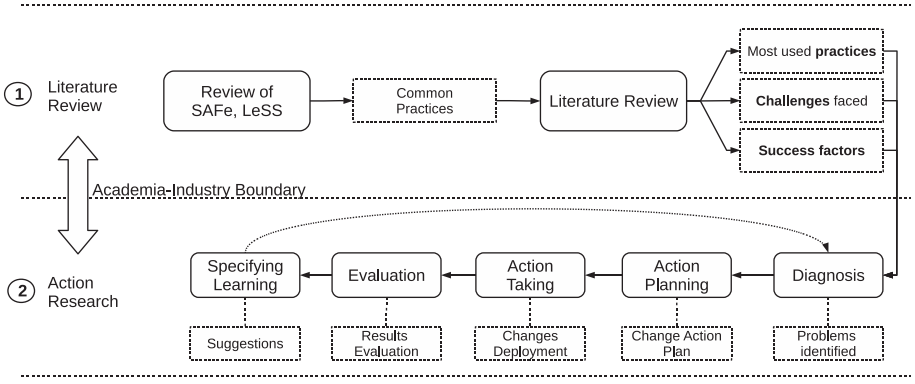

FIGURE 1 Research process from literature review to action research

and success factors in previous studies. We used similar process to systematic mapping studies (SMS) and systematic literature reviews (SLR),[41-43] although our aim was different and the applied process was less extensive. The main similarity was in mapping research outcomes, commonly done in SMS to map the current status of a research area,[42] to identify the quantity and type of research and results available.[41] In our case, the mapping was done between factors identified and previous research articles.

The second step was an action research process[37,38] (step 2, Figure 1) conducted by one of the authors during a process to scale-up agile processes within a software company. The identified practices, challenges, and success factors were the input for this process. Action research is a combination of theory and practice, as it attempts to provide practical values to the case organization while also acquiring new theoretical knowledge,[44] in an actionable and iterative manner.[45] Therefore, we iteratively provided feedback to the case company, while also observing the changes and analyzing them, in order to provide further knowledge about the large-scale agile implementation.

Action research has 5 identifiable phases that are iterated: diagnosis, action planning, action taking, evaluation, and specifying learning.[46,47] We conducted 2 loops of this cycle in our research. There were limitations with the action taking phase, as adjusting the whole development process is lengthy and our research was limited by time constraints: Some suggestions could not be implemented.

The diagnosis phase develops theoretical assumptions about the nature of the organization and its problem domain, with identification of the primary problems. This diagnosis was conducted with interviews with organization representatives presenting the organization's context and the problem domain for researchers.

In the action planning phase, an action plan that establishes the target for change and the approach to change is created. The proposed measures are guided by both the desired goal of the organization and the corresponding changes that would achieve the company's target. We presented the plan at consecutive meetings. The plan was in the form of concrete actions as well as additional suggestions that supported the change to the desired future state.

Once the action plan is executed, both the collaborative researchers and practitioners undertake an evaluation of results. The results were mainly presented by the representatives of the organization. As we mentioned, the action plan execution was limited by the ability to change the development process in constrained time.

The activity of specifying learning is formally described as the last phase of the action research method. However, it is usually ongoing process as the new knowledge is gained and reflected upon continuously.

3 | BACKGROUND—SCALING AGILE

When scaling agile to large scale, how can "large scale" be defined? There is no clear agreement in literature. While agile methodologies prescribe 7±2 developers per team, the discriminating factor to distinguish small- and large-scale projects is the number of teams, with 2 to 9 teams implying "large scale" and over 10 "very large scale."[48] A more restrictive definition of "large-scale" was given in Dikert et al,[34] considering 50 to 100 developers and 6 teams for a large development project.

There are nowadays several frameworks that can be used to guide the scaling process in organizations.[29] Based on the needs emerging over the years,[9,29,30,49,50] several methods, practices, and frameworks were derived to extend traditional agile methods: Scrum of Scrums (SoS), SAFe, LeSS, Disciplined Agile Delivery (DAD), Lean scalable Agility for Engineering (LeanSAFE), Recipes for Agile Governance in the Enterprise (RAGE)—with SoS, SAFe, and LeSS considered the more mature.[9,27,35,51]

According to the Version One 2017 survey on the state of agile,[56,57] the most used agile scaling methods are currently—ordered from the higher adoption level: the SAFe (28%), Scrum/SoS (27%), internally created methods (13%), lean management (4%), agile portfolio management (4%), LeSS (3%), DAD (1%), RAGE (1%), Nexus (1%). "Internally created" or ad hoc methods play a major role, combining several practices in new ways. We summarize in Table 1 the major frameworks for scaling agile and describe them in the following.

SAFe. Scaled Agile Framework is a framework as well as a collection of best practices of agile development for large enterprises.[35,36] The framework has been adopted by large enterprises such as Intel, Hewlett-Packard Enterprise, and Cisco.[27] The framework is built upon ideas of agile development, lean product development, and systems thinking. It supports companies of different sizes, from small ones with less than hundreds of employees to larger enterprises with thousands of people. To support such a degree of flexibility, SAFe has optional extensions for large companies.[35,36]

The SAFe definition is very detailed and sometimes contains even up to concrete agenda schedule of individual meetings.[35,36] Its organizational structure is large and has several hierarchical layers with many defined roles and responsibilities.[35,36]

The architecture of SAFe consists of levels. There are 3 base levels: Portfolio level, Program level, and Team level. For large enterprises, there is an optional Value Stream level. Across all the levels spans a Foundation level with additional elements that support organizations. Elements of the Foundation layer are, for example, SAFe core values, the lean-agile mindset, and SAFe principles.[35,36]

The Portfolio level is guiding the enterprise in its mission. It defines and governs the core strategic decisions that should bring value to the organization. The Program level then implements the defined mission. It manages, supports, and synchronizes multiple agile teams that create the solution. The Team level is the lowest level of SAFe. It describes how agile teams work: It uses Scrum, Kanban, and XP techniques in the development process of agile teams.[35,36]

TABLE 1 Comparison of major frameworks for scaling agile

Aspect	SAFe[35,36]	SoS[52]	LeSS[27]	DAD[53]	Nexus[54]	RAGE[55]
Team size	50-120 people in release trains	5-10 teams	10 Scrum teams, 7 members x team	200 people or more	3-9 Scrum teams	no specific size
Diffusion	High	High	Medium	Low	Low	Low
Maturity level	High	High	High	Medium	Low	Low
Complexity	High/Medium	Medium/low	Medium/low for Scrum-aware	High (many practices)	Medium/low for Scrum-aware	Medium/low for Scrum-aware
Organization Type	Traditional Enterprises	Traditional and Agile Enterprises	Large Enterprises	Multiple organizations & Enterprises	Traditional and Agile Enterprises	Traditional and Agile Enterprises

Source: Adapted from Ebert and Paasivaara[9] and Alqudah and Razali.[51]
Abbreviations: SAFe, Scaled Agile Framework; SoS, Scrum of Scrums; LeSS, Large-Scale Scrum; DAD, Disciplined Agile Delivery; RAGE, Recipes for Agile Governance in the Enterprise.

SoS. Scrum of Scrums is an approach to scale agile to projects with 5 to 10 teams, based on the reduction of communication overhead that can emerge when scaling projects.[52] The main idea is to designate some members of daily Scrum meetings as ambassadors that will participate to SoS meetings together with ambassadors from other teams.[58] Such daily meetings progress as normal Scrum meetings, with focus on challenges of coordination that might hinder team cooperation—all managed through a separate backlog.[58]

SoS is mostly a cross-team synchronization method, with the main goal of ensuring the quality of the Scrum processes in each team[52] but is often cited in studies as full framework for scaling agile.[9] However, it reaches full potential when adopted within the context of other frameworks (eg, SAFe[35,36]). In the remaining part of this article, we consider SoS as a scaling practice that can be used to scale-up agile processes.

LeSS. Large-Scale Scrum takes a different approach than SAFe.[27] Although still a framework, it is much more lightweight than the SAFe. The core idea of LeSS is that even for large organizations, there is no need for an overcomplicated process. LeSS tries to be compliant to one of the core principles of the agile manifesto—that individuals and interactions are more valuable than processes and tools. There are 2 variants: LeSS is for organizations that develop products that need up to 8 agile teams. LeSS Huge is for enterprises that require a higher number of teams.[27]

Compared with SAFe, LeSS is less strict in the description of the practices.[27] LeSS stays as agile as possible: focusing on mindset, values, and principles without introducing too many processes and roles.[27] The framework itself is much younger and does not have such enterprise support as SAFe.

Authors of both SAFe and LeSS frameworks agree that understanding the underlying values is important, and without their definition, processes are meaningless.[27,35,36] We can see some similar patterns in organizational structure and processes between these frameworks. Both frameworks use the scaled role of product owner and a hierarchy of backlogs. Moreover, they both lean towards feature teams[59]—cross-component, cross-functional teams—that can enable easier requirements management.[27,35,36] Furthermore, both frameworks use some additional teams that help the development teams with their work, the so-called undone department that can help with cross-cutting concerns such as quality assurance and their integration of the concept of "done" in a sprint/iteration.[27] Last but not least, LeSS and SAFe use scaled forms of common meetings, such as scaled planning, scaled demo, and scaled retrospective.[27,35,36]

DAD. Disciplined Agile Delivery extends Scrum by means of other agile methods such as lean and Kanban.[51,53,60] Many agile practices are part of the framework: Kanban for visualizing time-boxed work progress, Scrum, agile modeling, the unified process, extreme programming (XP), test-driven development, and agile data—agile strategies that can be applied to the "data" aspects of software systems implementation and deployment.[51,53]

DAD covers 3 main project life-cycle phases: inception (initial exploratory phase to understand project scope), construction (production of the solution), and transition (deployment of the solution).[51,53,60] DAD phases can be customized according to the needs: agile/basic (more similar to Scrum application), advanced/lean (more similar to Kanban), continuous delivery life cycle (brief transition period, ideal for quick releases, even daily), exploratory life cycle (for start-up/research projects with more phases for envisioning and measuring product releases).[51,53,60]

Apart traditional Scrum roles, DAD introduces a role called team lead (similar to Scrum master) and an architecture owner role derived from the agile modeling method.[51,53] DAD provided other roles to address scalability issues across teams (specialist, independent tester, domain expert, technical expert, and integrator).[51,53]

One difference of DAD compared with other frameworks is that it allows freedom of the teams to adapt the processes, and it also allows to be customized according the aforementioned life cycles, depending on the needs.[51] DAD is particularly useful for having guidance in the area of architecture and design and DevOps and can also be integrated with capability maturity models.[51] Nevertheless, it did not reach wide adoption so far.[51,56]

Nexus. Nexus was born with the intention of scaling-up Scrum practices, allowing for lower barrier to adoption and easier application.[51,54,61] Nexus allows to scale-up workload from 2 to 9 teams, with Scrum used to manage interdependencies across teams and problems that can arise when increasing the number.[51,61]

In Nexus, all Scrum teams work on a product backlog, aiming to reach an integrated increment with the Nexus sprint planning to coordinate the activities of all Scrum teams.[54] One product owner, a Scrum master, and an integration team member are part of an integration team in Nexus.[54] There are Nexus sprint retrospectives in which representatives from each Scrum team discuss sprint challenges. Such retrospectives are conducted in parallel with team retrospectives.[54,61]

Nexus is relatively less mature than the other frameworks; however, it is expected to reach wider adoption in upcoming years.[51]

RAGE. Recipes for Agile Governance in the Enterprise is a framework for agile governance, with processes applicable at the project, program, and portfolio level of any enterprise.[55] Key characteristics are rapid decision-making, light artifacts, and customization for different processes (fully agile, waterfall, and hybrid).[55]

At the project level, RAGE prescribes teams to be similar to Scrum teams with traditional activities such as sprint planning and retrospectives, with the governance level suggesting to apply ranking estimation and story development.[51,55] At the program level, area product owner and program manager conduct release planning meetings, SoS meeting, or release reviews, with the governance level focusing on release hand-offs, staging readiness reviews, product readiness review, and the validation of product development.[51,55] At the portfolio level, the portfolio owner, area product owner, and program managers conduct mainly portfolio grooming meetings to develop strategies, with the governance level focused mainly on monitoring, decisions, and quality assessments.[51,55]

RAGE allows quite a high level of customization of the processes but is relatively new, and the adoption level is still low.[51]

Merging the views. We used information from the frameworks to target the subsequent literature review. However, to simplify knowledge transfer within the company part of the action research process, we had to focus on a subset of frameworks—covering all of them with all small difference would have been unpractical and would have lead to more complex communication. We selected 2 main frameworks: the SAFe[35] and LeSS.[27] The reason is that these can be seen as 2 representative frameworks, as they take different approach to scalability: SAFe is the most complex agile scaling method, while LeSS attempts to be as minimalistic as possible: It relies less on processes and specific organizational structures and more on a mindset of people and ad hoc communication. For this reason, the upcoming sections take these 2 frameworks (and their practices) as reference frameworks for the agile scaling process.

We identified 8 common scaling practices that the 2 frameworks (SAFe and LeSS) utilize. We use for each of them an identifier (*SP = scaling practices*) to ease the readability:

SP1. Scrum of Scrums: approach to scale Scrum to large groups, dividing people into groups and having daily Scrums with ambassadors from the teams. We already covered SoS main characteristics, as it can be considered as a kind of stand-alone framework for scaling agile within an organization.[52] SoS is contained in both LeSS and SAFe.[27,35,36,52]

SP2. Communities of practice: a group of people with a common concern or passion that learn how to improve, share their knowledge, and expertise by interacting on an ongoing basis.[62] The main idea is that participation in such a community can be a way to increase the learning outcomes also at the organizational level.[62] Large agile organizations can benefit from various overlapping, informal cross-team communities,[63] and communities of practices can help in having leaders acting as coordinators of the community members, rather than acting purely as hierarchical superiors in agile projects.[64] Both LeSS and SAFe strongly suggest using communities of practice in a large-scale agile organization.[27,35,36]

SP3. Scaled sprint demo: a meeting in which teams show all the features that were implemented, with the focus on the main product under development.[27] SAFe calls it system demo[35,36]; LeSS calls it sprint review.[27]

SP4. Scaled requirements management: a group of scaling practices that are concerned about requirements management, such as the creation of the hierarchical structure of product owners, the hierarchical structure of product backlogs and its management. LeSS Huge uses requirement areas, area product owners, and area product backlogs.[27] In SAFe, this practice is utilized in the form of value streams, agile release trains, and its backlogs and managers.[35,36]

SP5. Scaled sprint planning: a meeting where the future of a product is discussed. SAFe calls the meeting PI planning,[35,36] and LeSS calls it sprint planning one.[27] The meeting is larger in the context of SAFe than LeSS. According to SAFe, the agenda of the meeting can be business context and product vision, architecture vision, organization readiness, and risk analysis.[35,36] LeSS suggests only a straightforward and short meeting, in which the product owner presents the prioritized list of items that are ready for development and their clarification together with the definition of the sprint goal. The teams then select which items they will work on.[27]

SP6. Scaled retrospective: a meeting that reflects on the execution of the previous iteration above the level of a single team. Teams should brainstorm about larger obstacles that are impeding them. The meeting is a part of the Inspect and Adapt workshop in SAFe.[35,36] LeSS calls it overall retrospective.[27]

SP7. Feature teams: cross-component, cross-functional, stable, long-lived, and self-organized teams that can help in designing, planning, and implementing in a sprint.[59] Therefore, a feature team should be able to fully implement a feature that spans across any component of a product, and ideally, the team should be stable for several years. LeSS and SAFe suggest using feature teams instead of specialized teams for individual components.[27,35,36]

SP8. Undone department: a group of teams that support development teams to achieve potentially shippable increments at the end of each iteration.[35,36] Both LeSS and SAFe agree that teams should be able to create potentially shippable increment at the end of each iteration.[27,35,36] An undone department is supposed to aid with the overall definition of "done" in an iteration. The undone department can support with all cross-cutting concerns that are part of product development, from architectural design to quality assurance (QA) activities.[27]

4 | REVIEW OF PRACTICES, CHALLENGES, AND SUCCESS FACTORS IN SCALING AGILE (STEP 1)

With the emergence of evidence-based software engineering,[65,66] a plethora of systematic literature reviews were performed in the area.[67] There is a huge variation of methods that can be applied in place or as complement of traditional literature reviews[68]: from SLR,[67,69] SMS,[41,42] multivocal literature reviews,[70] to meta-analysis reviews.[71] The main discriminating factor is the final goal of the review.[68] As reported (Section 2), our literature used similar process to an SMS and an SLR,[41-43] although our aim was different and the applied process was less extensive. In this section, we report about the focused literature review that was conducted as a first step towards the action research process.

4.1 | Methodology

The literature review study was performed using 4 digital repositories (*IEEE Explore*, *Springer*, *Research Gate*, and *ACM*). The goal was to map the practices identifies in the previous section from SAFe and LeSS to experiences and lessons learned in the transformation process. We aimed at answering 3 main questions:

RQ1. *Which agile scaling **practices** reported in literature did large companies (embracing practices from SAFe, LeSS) use?*
RQ2. *Which **challenges** reported in literature did large companies (embracing practices from SAFe, LeSS) face when adopting scaled agile development?*
RQ3. *Which **success factors** reported in literature helped large companies (embracing practices from SAFe, LeSS) to adopt and scale agile development?*

We defined search strings for 3 facets to find relevant primary studies:

1. Agile methods ("agile, lean, scrum, kanban, agile");
2. scaling frameworks ("large scale scrum, OR scaled agile framework OR safe OR disciplined agile delivery")
3. We could not include a third facet for large organizations, so we performed a successive filtering: To include one paper in the review, an examined company needed to have at least 50 employees and multiple teams developing one product.

We defined the following inclusion/exclusion criteria:

1. primary studies could be eligible;
2. inclusion of gray literature, if found (this was the main choice to include a repository such as ResearchGate)
3. English language only;
4. excluding papers that did not deal with the practices of SAFe or LeSS identified in the previous section;
5. excluding articles that did not provide lessons learnt in terms of evidence from case studies/surveys.

Results of full-text search in all the repositories (IEEE, SpringerLink, ACM, and ResearchGate) are presented in Table 2. After title and abstract reviewing, we included 12 articles in the review (Table 3) that also represents the mapping of articles in the figures to the corresponding citation.

TABLE 2 Repositories and query results

Repo	Query	Results
IEEE	*(agile, lean, scrum, kanban, agile) AND (large scale scrum OR scal* agile framework OR safe OR disciplined agile delivery)*	105
Springer	*agile AND lean AND scrum AND kanban AND agile AND ("large scale scrum" OR " scaled agile framework" OR safe OR " disciplined agile delivery")*	60
ACM	*"query":{ content.ftsec:(+agile, +lean, +scrum, +kanban, +agile large scale scrum, scaled agile framework, safe, disciplined agile delivery) }*	73
Research-Gate	*manually queried*	–

All were full-text search (not just meta-data).

TABLE 3 Mapping of articles to references

ID	C1	C2	C3	C4	C5	C6	C7
Ref →	Vallon et al[72]	Moe[73]	Paasivaara and Lassenius[74]	Long and Starr[75]	Schnitter and Mackert[76]	Atlas[77]	Hanly et al[78]
ID	**C8**	**C9**	**C10**	**C11**	**C12**		
Ref →	Paasivaara et al[79]	Paasivaara et al[80]	Schatz and Abdelshafi[81]	Hajjdiab et al[82]	Benefield[83]		

4.2 | Results

To answer the research questions, we next present 3 mappings (practices, challenges, and success factors), discussing each finding according to the classification scheme that was derived from either the articles or the initial review of scaling frameworks. If some papers are not included in one of the mappings in subsequent figures (Figures 2, 3, and 4), it is because we did not find a relevant mapping to the factors in the paper.

4.2.1 | Scaling practices

Taking into account the common practices identified in the SAFe and LeSS frameworks, this section describes which scaling practices were found in the reviewed literature (Figure 2).

SP1. Scrum of Scrums: was the most used scaling practice.[72,74-76,78,79] A typical use of SoS was for synchronization between teams,[72,74,76] but in some cases, it was used as a daily meeting, which can lead to problems over time due to the many distributed teams that needed to be represented.[74,79] Feature specific SoS was used to avoid the problem of too large meetings.[79] Oversize SoS meetings can cause teams to stop reporting their impediments, feeling there is not enough time during the meetings.[74] As LeSS or SAFe suggest, having team representatives in the meetings can reduce this issue; however, they need to be involved in team's work to bring useful points to the discussion.[27,35,36]

SP2. Communities of practice: In total, 5 cases mentioned the use of CoPs.[75,77,79,80,83] CoPs resolved the lack of information about system design, agile development, and tools[77,79] or were used for knowledge sharing and promotion of best practices to facilitate the transition to feature teams.[80] CoPs were also used for coaching, coordination, and design between teams, improving software craftsmanship and technologies and improving the way of working.[84]

SP3. Scaled sprint demo: In all cases, the scaled sprint demo was utilized as a demonstration of features of all areas of a product.[72,74,75] Scalability of the demo can be an issue in case of many teams attending,[74] suggesting to either let the product owner attend the demo sessions or using a science fair model with teams showcasing their effort on some stands.[74,75] The case by Paasivaara and Lassenius[74] experienced several problems with the scaled sprint demo. At first, the meeting was held in a big auditorium with all teams attending it. As the project grew, the scaled sprint demo became too large. Thus, teams started to have separate demo meetings, with only product owners attending them, but this did not give teams enough context of the whole product. The next step was to use the scaled sprint demo in one place with all the teams again, but due to time constraints, the demonstrations of the teams were reduced to slide presentations. Authors reported that the organization is planning to revert the scaled sprint demo back to separate demonstrations of individual teams.

The organization reported by Long and Starr[75] went through a similar transformation of scaled sprint demo as the previous organization.[74] The first attempt was to let teams do a demonstration of their work in one place. The second stage was to have only a slide presentation by each team. The last state of the meeting was a so-called science fair model: "each team sets up a display, and employees visit those displays they find interesting. Each 'booth' offers a 15-minute presentation about recent work efforts."[74] A similar model is suggested by LeSS.[27]

SP4. Scaled requirements management: Scaling of requirements management was identified in 3 cases.[74,76,80] Cross-component dependencies can be an issue as they lead to large component overhead not allowing separate meetings for different product areas.[74,76] Adjusting the architecture of a product according to the needs of large-scale software development can be a key factor.[76]

SP5. Scaled sprint planning: Scaled sprint planning was identified in 3 cases.[72,74,80] It might be challenging in a distributed project with multiple sites, as it can lead to late planning, wrong estimations, and misunderstanding of requirements.[72] Additional meetings were required to explain requirements to the teams sufficiently.[74] Teams also needed more detailed explanation and clarifications in terms of Feature Investigation and Feature Concept Study for each feature, to find out if a feature is doable, how much effort is required, and how it can be implemented.[80]

SP6. Scaled retrospective: Scaled retrospectives were identified in 2 cases,[72,74] noting that problems identified can be too large and complex to be solved. An issue with scaled retrospective in the organization reported in Paasivaara and Lassenius[74] was that the problems identified in the scaled retrospective were often too big and hard to solve. Resolution of these challenges might take several months. Therefore, the same problems were identified by teams in several consecutive iterations of the meeting. Thus, teams questioned the usefulness of the meeting as they identified the same impediments again and again, without any progress.

SP7. Feature teams: Only 2 cases[79,80] mentioned the use of feature teams. Unfortunately, none of these cases was successful. In one case by Paasivaara et al,[80] the organization created cross-component, cross-functional teams specializing on specific business flows. This specialization

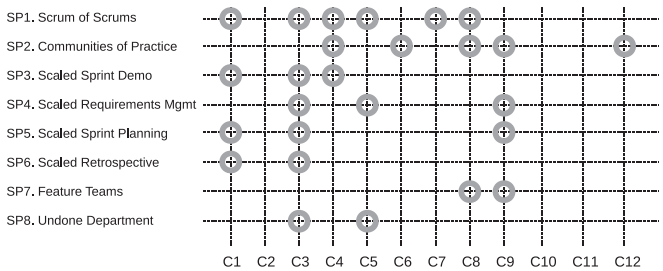

FIGURE 2 Mapping of papers to individual scaling practices. O = The practice is discussed in the article

allowed the teams to focus on a narrower product area.[80] In another case by Paasivaara et al,[79] the solution was to establish a systematic exchange program between teams.

SP8. Undone department: although many cases had to cope with architecture design problems and with inadequate development infrastructure,[72,74,75,79] only 2 cases mentioned the use of an undone department,[74,76] namely, an architecture team consisting of 9 high-level technical specialists in Paasivaara and Lassenius[74] and testers, quality assurance experts, and DevOps teams in Schnitter and Mackert.[76]

4.2.2 | Challenges

There were several challenges that the companies experienced during their transformation to large-scale agile development (Figure 3). We summarize them, together with comments about each of the challenges. For each challenge, we give a common identifier (*SC* = *Scaling Challenge*).

SC1. Resistance to change: resistance to change and attachment to previous processes were very common challenges. Some form of resistance to change was reported in 8 papers.[73-77,80,82,83] The resistance occurred at all levels of organizations, including development teams, middle and upper management. Moreover, in large organizations, the resistance to change on higher levels, such as middle and upper management, is a more significant problem.[73,74,77,80,82] For a successful transition, a supportive change management is necessary: Several cases faced significant challenges due to lack of it.

One of the reasons why team members did not want to change to agile development was an increased transparency[75,76] because people felt observed and did not want to share their problems. Another reason for the resistance were the new responsibilities that agile development brought to teams. Teams are expected to be self-managed in agile development, but not everybody was pleased. As Long and Starr[75] reported, teams did not want to solve their new problems. Firstly, in agile development, middle management is not expected to manage teams anymore. This shift of responsibilities of middle management caused confusion as well as resistance to change because managers feared that they would not be needed anymore.[73] Secondly, managers did not respect team's decisions and their visions of a development process. In Paasivaara and Lassenius,[74] managers tried to micromanage teams. This micromanagement caused several problems, one of them was that teams lost interest in meetings and stopped attending them, as they were not responsible for synchronization and communication and therefore, found the meetings useless. Lastly, in Hajjdiab et al,[82] teams were unmotivated because of lack of resources and lack of recognition for their efforts to improve the development process. In the end, the teams gave up their endeavors.

SC2. Distributed environment: Any organization that had its development teams distributed across several sites experienced significant challenges with agile development. Challenges due to distributed environment were reported in 6 cases.[72,73,75,76,79,80] Agile development puts emphasis on constant communication and team spirit. Distributed environment makes these close relationships hard to obtain.[76] Transparency is one of the cornerstones of agile development, and without it, agile development cannot succeed. Achieving transparency between teams that are distributed across multiple sites is very complicated.[72]

Information sharing might be easily forgotten in a multisite environment. Information from discussions and meeting notes from one site might not reach other locations. The organization should emphasize transparency of all information between sites and provide necessary tools, such as wikis and video conferencing. It is important to have all the needed knowledge and infrastructure at each location.[79] To mitigate these challenges, Paasivaara et al[80] suggested visiting engineers and providing high-quality video conferencing equipment.

SC3. Quality assurance issues: Five studies[73,75,76,79,80] reported problems with quality assurance and a loss of quality in code shortly after the transition to scaled agile processes. The quality loss was caused mainly by additional responsibilities, pressure on teams, or bad definition of the concept of "done."[73] Teams skipped bug fixing[73,75] and testing and did not use continuous integration[79]; thus, technical debt piled up. Problems with large technical debt might take years to solve.[79] The quality loss resulted into teams discouragement. The teams started to lie and report false status of their work to hide the problems.[73] These issues were mainly solved by introducing an undone department and training. For example, the organization reported in Long and Starr[75] included testers and quality assurance experts in the teams. The experts defined tolerance levels of technical debt for individual products: New features could not be added to a product whose tolerance levels were not satisfied. Additionally, Long and Starr[75] proposed that teams should be fixing bugs immediately. Paasivaara et al[80] suggested establishing a common backlog, introducing supportive roles, such as people responsible for a particular subsystem and architects, building an appropriate

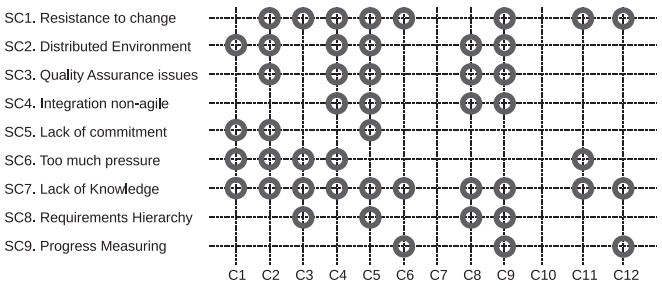

FIGURE 3 Mapping of papers to individual challenges. O = The challenge is discussed in the article

test environment, and training personnel on continuous integration. The undone department of the organization[76] included an architectural team, a support team—which removed impediments that could not be resolved by development teams—and an expert lean team.

SC4. Integration with nonagile parts of organization: Misalignment of organizational structures caused problems in 4 cases.[75,76,79,80] For example, the problem was that nonagile teams did not want to rely on agile teams—not knowing whether the agile teams would deliver their work on time. These issues can be solved by including waterfall parts of the organization in the planning process and involving nonagile teams early in planning process.[80] Also, improvement of continuous integration and test automation systems helps, as it allows faster and better integration.[80]

SC5. Lack of commitment and teamwork: Three cases reported problems with commitment and teamwork.[72,73,76] Vallon et al[72] indicated that issues with the dedication of teams were caused by frequent changes to user stories that were being implemented and uncertainty due to wrong and late specification of user stories. Lack of teamwork was also due to the specialization of team members.[73] Specialized people in teams were concerned with their problems and did not want to commit to a common team plan. Team's definition of "done" was also very different across team members: Individual team members prioritized different concerns based on their specialization. Distributed decision making was hindered as well; team members did not understand work of others and therefore did not want to rely on the others. Additionally, knowledge sharing was not working as the specialists realized that they are important for the company. They did not want other developers to know how to do their special job because that made them more indispensable for the company. All of this resulted in a very competitive environment, and therefore, some people were scared to report their problems. In one case, teamwork was hindered, because people thought that Scrum was introduced to increase efficiency and substitutability of programmers. Also, they thought that flatter management hierarchy offered fewer career opportunities. Hence, people tried to be more visible and competitive.[76]

SC6. Too much pressure and workload: High pressure and workload caused many problems. Issues related to them were reported in 5 cases.[72-75,82] A new way of working, new responsibilities, together with constant market pressure increased tension in teams. This tension resulted in bad quality code and late planning.[72] Teams also skipped testing, refactoring, defect fixing, and engineering practices, thus creating a lot of technical debt.[73,75] Furthermore, as the teams that were new to agile development had not yet realized the importance of process improvement, too much pressure caused them not to comply with their responsibilities.[73,74,82] Time deficit caused teams not to care about the identified issues from their retrospectives. Hence, these problems never made to the top of their backlogs.[73] In Paasivaara and Lassenius,[74] inexperienced teams did not have a shared vision of Scrum and their development process yet; therefore, it was essential to let them have enough time to develop such vision. Moreover, high pressure impeded communication in teams. Teams started to skip and postpone meetings. Therefore, daily meetings became weekly and then even monthly.[82]

SC7. Lack of knowledge, coaching, and training: The most common challenges—which in many cases became the cause of other difficulties and problems—were a lack of knowledge, training, coaching, or understanding. These issues were reported in 10 cases.[72-77,79,80,82,83] Several factors caused the lack of coaching and knowledge, most of the time due to underestimated difficulty of the agile transformation, financial constraints, lack of support of management, or rushed transition. For example, the organization in Hajjdiab et al[82] underestimated the difficulty of agile adoption. The organization did not have sufficient knowledge in their ranks but refused to use external expert because of financial constraints. Instead, managers who were new to agile took the role of agile experts. There were several consequences of lack of coaching and training. Teams could not change their habits as easily as they expected. Schnitter and Mackert[76] reported that because of a fast transition with a lack of knowledge and experience, many assumptions were purely theoretic. Therefore, a lot of unexpected problems emerged. Additionally, the case reported that the teams that were mentored performed better than the teams without mentoring.[76]

The consequence of lack of knowledge was the erroneous implementation of agile practices. For example, in 3 cases, people did not understand the purpose of meetings.[73-75] Teams were not finding underlying causes in retrospectives but merely identifying symptoms. Also, teams did not respect core principles of practices they were supposed to implement, such as respect for iterations in Scrum, prioritization of user stories, or the importance of customer contact.[74,75]

The consequence of such misunderstanding was an overzealous change. Although the teams had all the information about the practices, they did not understand them and used them improperly. For example, the organization in Benefield[83] experienced a situation where a manager enforced Scrum practices in a strict, by-the-book fashion and forced them against their will. Also, some teams followed the practices too religiously and lost perspective.

Furthermore, the transition to large-scale agile required training in domain knowledge as well. In Paasivaara et al,[79] the organization had to train teams in other domain areas to establish feature teams, because of the previous component-based architecture of their product. Appropriate coaching of management increased management support of agile methods.[83]

SC8. Requirements management hierarchy: In large projects, requirements are not manageable by a single person (product owner); hence, there is a need to divide the responsibility of their management. This division was in many cases a significant challenge.[74,76,79,80] Scaling of requirements management cannot be avoided when scaling agile. The fact that some organizations had to adjust the whole architecture of their products, like in Schnitter and Mackert,[76] shows such difficulty. The fact that the entire transition may fail because of the wrong division of a product in areas as in Paasivaara and Lassenius[74] demonstrates the importance. Hence, it should be done carefully and properly.

SC9. Measuring progress: When an organization is changing, the development process, measuring the changing trend, is important. However, this proved to be challenging in large-scale agile development, as 3 cases[77,80,83] experienced difficulties when it came to monitoring and measuring the progress of their agile development. The biggest problem was finding the right metrics—the organizations did not know what to measure to get meaningful results.[83]

4.2.3 | Success factors

From all the papers included in the review, we identified and grouped reported success factors into 7 groups (Figure 4). Each factor is given a unique identifier (*SF = Success Factor*).

SF1. Acquire knowledge: As the most reported challenge was a lack of knowledge, it is no surprise that the most common reported success factor was to try to increase the level of knowledge and expertise on agile practices. Thorough, deep, and systematic knowledge acquisition and sharing were stated as success factors in 11 cases.[72,73,75-83] The most recommended way to acquire knowledge and expertise was to hire an external expert with broad and deep familiarity with agile development.[75-78,82,83]

The external expert shared his knowledge with several internal employees, who then spread it across the organization. One of the reasons why it was more beneficial to bring an expert than investing into knowledge acquisition from other sources, was that organizations realized the general and theoretical suggestions were not easily applicable in their particular context, and a deeper understanding of the methods was needed.[72,78] Another recommendation was to coach people deeply and not broadly. For example, Hanly et al[78] stated that coaching has to be about both values and processes, but unfortunately, the values are often omitted. The lack of understanding of values might lead to a lack of pragmatism and inability to deal with the new and unknown situations. Agile means to be flexible, but its tailoring requires a deep understanding of its values. It is better if people are committed to constant learning than committed to particular agile method or principle.[73] Also, it is valuable to teach teams how to learn, for example, using double-loop training, so they are better at finding an underlying cause of a problem, and they can adjust their team processes correctly.[73]

According to 2 cases,[77,83] it is important to coach people and not to dictate them. Coaches should be using "should" instead of "must" and let the transformation be a bottom-up approach. As for successful agile transformations, it is a must to have employee support.[83] Furthermore, a systematic training of coaches and other people inside of an organization was required for successful large-scale agile adoption.[77,78,82,83] Systematic coaching and training both mean to have recurring events, presentations of external experts or communities of practice. Besides, systematic training ensures enough available coaches in an organization. Lack of coaches leads to the uncontrolled and untrained adoption of agile methods by overzealous teams. Changing their wrong habits was harder than changing a team without any experience.[83] Moreover, so-called pair coaching, which combines agile experts with an expert with domain knowledge to enable more efficient and accurate coaching, proved to work well.[78,83]

SF2. United view on values and practices: For a successful agile transition, it is necessary to define a common view on the change. In total, 6 cases found unification of values, definitions, way of working, and understanding beneficial.[75,76,78-81]

There is a need to define roles, their responsibilities, and common definitions properly. Also, it is important to point out wrong ideas and misconceptions.[78] Teams are more willing to follow an unified view on Scrum.[75] Schnitter and Mackert[76] stated that it was beneficial to use a common language between teams, architects and product owners, such as established agile definitions, the unified modeling language or fundamental modeling concepts. There are several ways to establish a shared view and way of working. For example, it can be done through coaching, documentation,[78] communities of practice,[84] or value workshops.[80]

SF3. Tools and infrastructure: Organizations have to be prepared to provide sufficient resources to teams while transforming to agile development. Five cases[72,76,79-81] stated that decent tools and infrastructure were helpful. The tools include, for example, development and local test environments, continuous integration and automated tests, and communication infrastructure including video conference equipment.[79] Common tools and infrastructure were particularly beneficial in a distributed environment, where teams need all the help to reduce their communication impediments.[72]

SF4. Solid engineering practices: Many teams experienced loss of quality in their code during an agile transformation. Therefore, having solid engineering practices is crucial.[75,83] Long and Starr[75] suggested including QA and testers into development teams. Additionally, good technical debt management, such as tolerance levels and visual indications for the tolerance levels, was suggested.[75]

SF5. Careful transformation: Changing an organization to agile development means changing the mindset of people. This change might take a long time. Hence, a slow and careful transformation was needed in 6 cases.[73,75,76,81-83] It is better to perceive the transformation as a long-term organizational change.[73] The organization should improve the development process gradually and take at least 3 months before changing methods and practices.[75] Organizations should also reassure to have all the necessary resources—such as enough time, a prepared plan, and

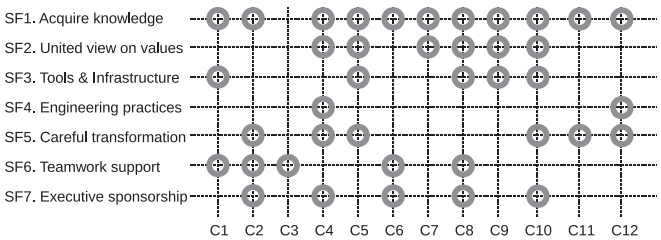

FIGURE 4 Mapping of papers to individual success factors. O = the success factor is discussed in the article

expertise—before making the transition. Teams need time and space to learn and adapt to the new way of working and change their habits. Thus, the organization should give them enough time and not overpressure them[82] or let them work overtime.[81]

SF6. Teamwork support: Seven cases reported that close connections and constant communication between teams and team members are necessary for successful agile development.[72-74,77,79,81,83] The organization should establish a transparent environment for openness in the team without fear of discussing problems to improve teamwork.[73,83] Also, it is better to keep teams small.[77] Communication between teams and team members can be enhanced by SoS meetings and CoPs. These practices are "both forums and provokers of discussion."[79]

SF7. Executive sponsorship: Executive sponsorship is the most fundamental factor for a successful transformation of an organization.[73,75] Without proper executive sponsorship, no other success factors would be realizable, as not enough resources would be allocated. Five cases stated that an executive sponsorship and management support is a must.[73,75,77,79,81] Obtaining executive sponsorship might be problematic if the transition to agile is a bottom-up incentive. The only suggestion we found was to gather enough proof about the effectiveness of agile development, by means of a measurement program.[73,75,77,79,81]

4.3 | Threats to validity

There are few internal and external limitations in the literature review performed that need to be considered. We did not follow all the phases of the systematic mapping processes,[41,42] as our aims were different. Concretely, we did not conduct the fourth phase—keywording of abstracts. However, there were 2 reasons for this decision. Firstly, we already had a set of categories of common scaling practices from the initial definition of practices. Secondly, success factors and challenges cannot be inferred by simple keywording of abstracts.

The needs for our literature review were peculiar because of the collaboration with industry. It is common knowledge that most of the practitioners do not read research articles.[85] This does not depend—or depends partially—on the fact that research is answering interesting questions from practitioners' point of view, as there can be a gap,[86] but also from the fact that that journal and conference papers are *not* a good communication channel between academia and industry.[87] For this reason, we needed some targeted review for the company audience: providing to company's stakeholders previous excellent systematic reviews published in the area (eg, Dikert et al[34]) would have not been sufficient to reach our goals. Thus, the literature review in this paper needs to be read in this sense, not as exhaustive, rather supportive of the subsequent action research process. Our goals were to provide the organization with the synthesis of all was known from literature in terms of mapping of practices from SAFe and LeSS. The extensiveness of the review was not the main goal: The completion of the review was needed in due time before the action research process could start, as collaboration with industry need to comply to strict time requirements.[88-90]

5 | ACTION RESEARCH IN A LARGE ORGANIZATION (STEP 2)

The review of the practices, challenges, and success factors from literature was the theoretical input for an action research run within Kentico,* a software development company with headquarters in Brno, Czech Republic. Kentico was founded in 2004 and has nowadays around 200 employees. The primary product is a platform that includes a content management system and e-commerce and online marketing features. Recently, the company started to shift its focus towards the Cloud platform. The project started with one team that was given absolute freedom to define its development process. Other 3 teams were added. These teams were still free to select tools and technologies and partially define their development process. The teams were separated from the former organization structure. Each team developed a different part of the complete Cloud solution. The main problem was that the Cloud platform started to grow, and the organization wanted to allocate more developers and resources. Therefore, a more precise development process was needed.

5.1 | Contextual information

At the beginning of the action research process, the new Cloud platform had 4 development teams (Figure 5). Each team had its product manager and agile coach. product managers were coordinated by the director of product and agile coaches by the development director. The architecture team was being consulted regarding the product architecture. There were additional specialized roles that supported the teams: There was a security specialist who specialized in product security, and quality assurance coordinator who was in charge of the DevOps team. Agile coaches, the security specialist, and the quality coordinator were coordinated by the development director. There was also a team of technical writers that was responsible for the technical documentation. Furthermore, a user experience team was available. We describe in more detail the different roles, as the action research process started.

*www.kentico.com

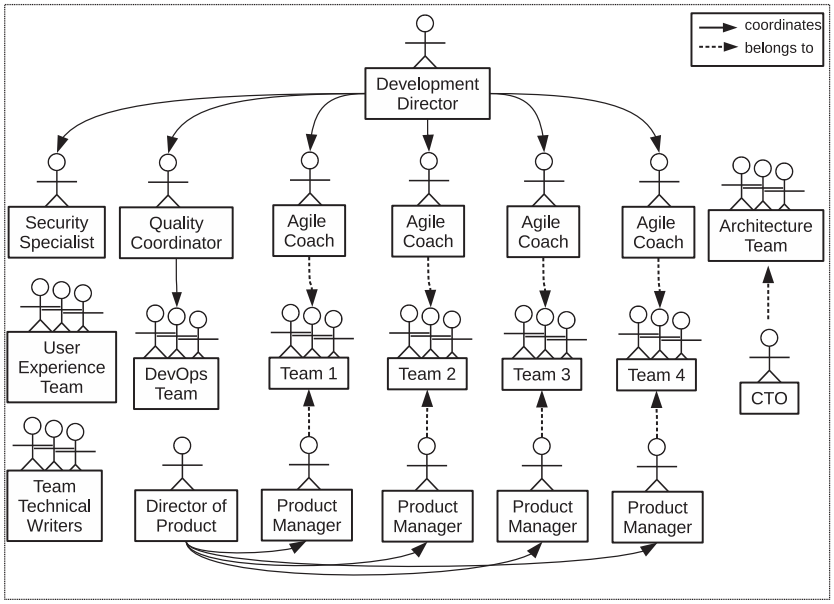

FIGURE 5 Structure of the organization

5.1.1 | Development teams

At the beginning of the action research process, all 4 development teams were using Scrum. Additionally, lean principles were utilized, mainly focusing on waste elimination, but also addressing other 6 principles of lean. Each team had its product manager and agile coach. Moreover, some team members had specialized roles such as DevOps and UX designers.

Teams were in contact with the customer primary through interviews aimed at solution validation. Usually, one developer together with the product manager was communicating with the customer. Other team members could use recordings of the calls. The product manager was summarizing the result from the interviews on a daily basis. Additionally, everybody in the teams had access to the so-called product board, which was a tool used for aggregation of feedback from customers.

Teams had a synchronization meeting every day—the Scrum daily meeting. Every week, there was the whole Cloud platform synchronization meeting. It was attended by the director of product, the CEO, and representatives from all teams. Additionally, each team had a meeting with the CEO to discuss their progress and future steps. The other meetings such as design workshops, synchronization meetings between teams, and risk analysis meetings were organized ad hoc.

In 3 teams, the backlog was owned by the team instead of the product manager. The reason was to allow product managers to focus more on their tasks. Initially, the backlog was prepared by the product manager, but the company decided that the role should concentrate more on obtaining feedback from customers and on market analysis. Thus, the ownership of the backlog was moved to the teams while product managers were bringing inputs to the teams from outside. This change helped the organization as intended. Product managers had more time for focusing on the market. Additionally, the teams better understood needs of customers, since they needed to work with the previously provided inputs. Therefore the teams were more active, asked additional questions and analyzed data more carefully. There were also some challenges connected to this change. Firstly, at the beginning, the motivation behind this change had to be explained properly, so the teams would participate. Secondly, the teams were in different stages of product development, thus when a new feature was being implemented, the overall view-point and its implementation were very different in the individual teams. Last but not least, the new way of working created additional overhead for the teams for the creation of the user stories.

5.1.2 | Product managers

Product managers were responsible for finding a problem-solution fit and product-market fit, the return on investment, the competitive landscape analysis, the win/loss analysis, the buyer and user personas, and the go-to-market strategy. The responsibility for the go-to-market strategy was shared with sales and marketing departments. Product managers were supervised by the director of product.

Product managers were part of the development teams: They were sitting in the same office and communicated with the teams on a daily basis. Synchronization between product managers was mostly ad hoc. Two synchronization meetings were held each week, one for cross-team synchronization and the other for synchronization among agile coaches and product managers.

5.1.3 | Agile coaches

Each team had one agile coach. The 4 agile coaches together with the development director were part of the Scrum master team. Agile coaches were responsible for the satisfaction of team members, their motivation, team organization, and team administration, information spreading, helping teams with global issues and impediments, and soft skills teaching. They also participated in the hiring process.

Agile coaches increased their skill by sharing their knowledge, for example, using knowledge base tools such as the confluence software and attending agile and lean conferences. Agile coaches also visited other companies and shared their experience with them. There was also the so-called Scrum Club, which was a 1-hour meeting for additional knowledge sharing.

Agile coaches attended several meetings such as SoS, Cloud platform synchronization meetings, and product strategy meetings. They were in contact with the teams on a daily basis and also had regular one-to-one meetings with team members.

5.1.4 | Director of product

The director of product was responsible for the complete accomplishment of the product. His responsibilities included timely product delivery to the market, product communication to the market, and product adoption by the market. He was also driving cross-team prioritization and validation of cross-team iteration outcome, by managing product managers. He attended weekly teams synchronization meeting and weekly synchronization meetings for agile coaches and product managers. Additionally, he was having one-to-one meetings with product managers.

5.1.5 | Development director

The development director was responsible for the effective cooperation among the development and other departments, correct operation of the development organization, increasing employee value and helping them with career paths, correctly interpreted and implemented company strategy and vision, performance and efficiency growth of development, and implementation of continuous improvement mechanism into the development organization. The development director visited synchronization meetings for the Cloud platform, agile coaches, and product managers every week.

5.1.6 | CTO and architecture

The CTO was a member of the architecture team. His role was to consult the architecture of the planned features. He was also a team member of one of the development teams. The architecture team reviewed the implemented user stories from the architectural point of view and notified the teams if there was something that needed to be addressed.

The architecture was discussed during the Cloud platform synchronization meeting and during ad hoc meetings when the teams needed to discuss.

5.1.7 | DevOps

The DevOps team was responsible for various tools and the development infrastructure: This included version control tools, code merging, monitoring applications performance, infrastructure configuration and management, release automation, artifact repository, test and result performance, continuous integration tools, and build status. DevOps representatives attended synchronization meetings for agile coaches and SoS. Additionally, there were weekly synchronization DevOps meetings and also ad hoc DevOps meetings with the development teams.

5.2 | Research questions

We defined 3 main research questions that were the target for the whole action research process. These research questions were mimicking the ones we set for the initial literature review part (Section 4)—this time contextualized to the main company's context. This would allow to determine differences between the case studies included in the literature review and the results derived from the action research.

RQ4. *Which agile scaling **practices** were the most effective in the company's context?*
RQ5. *Which **challenges** did the company face when adopting a scaled agile development process?*
RQ6. *Which **success factors** were relevant in the company to adopt a scaled agile development process?*

5.3 | Data collection and process

The collaboration with Kentico lasted for a period of around 8 months, in which the action research process took place. During this period, one of the researchers was in contact with Kentico and was actively involved in the provision of feedback based on the necessities arising from the scaling process (Figure 6). The researcher had access to the online tools used for documenting the development process—mainly issue trackers—with a more limited set of permissions than developers. He could also have access to meeting notes—when the information shared during the meetings was not considered as reserved. Another author was the main contact at Kentico and would allow for faster communication with the development teams and the other parts of the organization. The main contact point was helpful to reduce the burden of communication and provide access to any relevant information, granting that no reserved information would leave the company. If information from the teams—that could not be derived from the shared documents—was needed, the main researcher would get in touch with the main contact point at Kentico. The same worked in the other direction, from the development teams to the researcher, as a kind of 2-way communication. Although this seems as a level if indirection, in

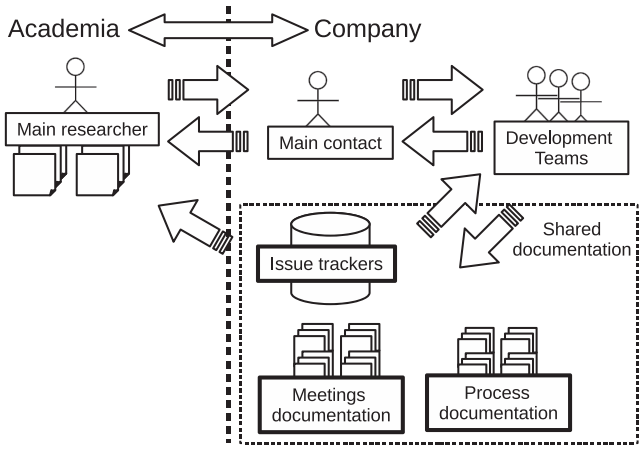

FIGURE 6 Collaboration academia-industry during the action research process (structure of the development teams is explicated in Figure 5)

reality, it provided quick information to the researchers. Only violation for this research setting were direct interviews from the main researcher to different roles in the development teams that clearly could not pass through the main contact point at Kentico. Based on the literature review and emerging new knowledge during the process, suggested practices were communicated by the main researcher to main contact point that would then provide feedback back to the researcher based on the application within the teams. Clearly, the changes to the development process were at the discretion of the development directors/development team—this was a real setting in which products needed to be shipped.

Data was collected/exchanged continuously using several methods.[†]

1. There were several meetings during which the problem domain at Kentico was defined.
2. Afterwards, the main Researcher presented solutions from literature in several iterations and discussed the progression with concrete problems in more detail.
3. Once the problems at Kentico were properly understood, a questionnaire was created. The questionnaire was given to the main contact at Kentico that distributed it to the different roles. The roles that we included were managers, developers, agile coaches, DevOps, and members of support team. The goal of the questionnaire was to get additional opinions from different points of view. Additional questions were discussed using online communication. The questionnaire was submitted twice, once at the beginning of the research process and at the end.
4. Interviews were the primary source of information. The main researcher conducted 6 interviews, which were about an hour to an hour and a half long. Additionally, constant online communication was realized for additional specification, clarification, and validation of information.

There can be different types of action research,[92,93] and the type of action research we applied was in line with action science: where the goal of research is to solve a problem in client organization by exposing differences between given theory and theory in use.[94]

The design of the action research process took inspiration from the dialogical action research process,[91] in which knowledge evolves over time from both sides of researchers and practitioners (Figure 7). The main researcher would propose to the main contact at Kentico approaches to experiment based on the evolving knowledge from both the literature review and the previous feedback from the development teams. Development teams would get information about approaches that could be meaningful to try in a real setting. Iteration zero dealt with the provision of the results from the literature review (Section 4). The diagrams from the literature review (Figures 2, 3, and 4) proved to be a quick and effective way of communication also with the main contact point. Such knowledge evolved during the action research process, as some of the approaches were changed by the development teams.

5.4 | Results

We present the main results from the action research, discussing changes during the process (Section 5.4.1), the identified scaling practices (Section 5.4.2), challenges (Section 5.4.3), and success factors (Section 5.4.4). We conclude with suggestions that were derived from the action research process (Section 5.4.5).

5.4.1 | Changes during the action research process

Compared with the initial situation at Kentico, we reported that at the beginning of this section, there were several changes that were performed.

- **Meetings.** Scrum daily meetings were still used within the development teams. In general, the synchronization of the teams via various platforms stayed the same (no change). A leadership meeting was introduced to discuss company topics (mission, vision, and strategy).

[†]Questionnaire and interview material are available online https://goo.gl/n81tAj

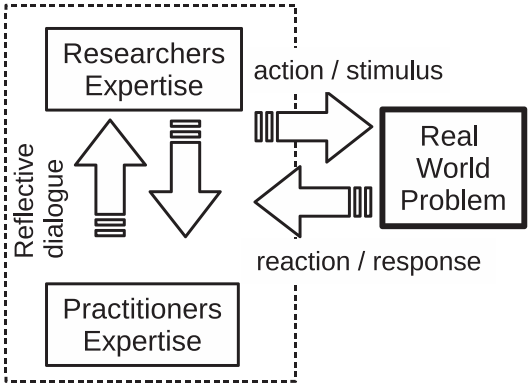

FIGURE 7 The dialogical action research process (adapted from Mårtensson and Lee[91])

- **Backlog Management.** There were no changes to the product backlog management. Opportunities (Epics) defined by product managers and UX teams were introduced.
- **Changes in roles.** The development director is now closely cooperating with the Scrum master leader for assistance in solving strategic topics. The product manager team lead role was introduced—helping with the organization of the product manager team (similar to a Scrum master leader role).
- **Software architecture.** Changes introduced that every team has to validate their changes in the architecture with the architect role (currently 2). The team has to prepare the overview regarding the opportunity and its effect on the architecture; architects will then review it and discuss with teams further questions if needed.
- **Scaled retrospective.** The milestone retrospective (held 4 times a year) was introduced—people in the development can add their topics, and after all topics are collected, there is voting mechanism. Top voted issues are then discussed in separate meetings—all voters are invited to the session.
- **Scaled planning.** There is more flexibility—originally the prioritization was happening every 3 months only. The prioritization happens now on monthly bases, where new opportunities are introduced and then prioritized against already prioritized ones.
- **Iteration reviews.** Shifted to Thursdays biweekly—2 parts: public, and private (where news related to marketing, sales, and customer success are shared).
- **Scrum of Scrums.** The concept remained the same.
- **Requirements management scaling.** The handling of the backlog was unified across the teams. There is one backlog with all the bugs for cross-development teams (based on the agreed rules, the teams are supposed to take specific amount of bugs in one sprint and fix them). There is also one common backlog for UX improvements. Each iteration has a naming convention for automation of data collection: status from iterations can be automatically filled daily without manual operation.
- **Undone department.** The concept remained the same.
- **Communities of practice (guilds).** No significant changes. Coding Dojos[95]—meetings in which group of programmers get together to learn and practice—were run monthly.
- **Differences in agile culture across company.** The new vision and mission was discussed and introduced to the employees. A new aspirational value was introduced—"Strive for excellence." In general, continuously visiting conferences, gatherings, and sharing with other companies the experiences, being open to any practice, are relevant aspects.
- **Process measures.** Continuously monitoring the backlog readiness and the ratio of maintenance within every iteration is applied. Committed/delivered is also being monitored.
- **New software engineering practices.** Usage of the Gherkin notation[‡] was introduced to be able to better address the need of automation tests. Combining agile, lean methodologies and experimenting are aspects that are enforced.

5.4.2 | Scaling practices

Throughout the action research process, we identified a series of agile scaling practices that the organization used. The identified "7 scaling practices" were SoS (SP1), communities of practice (SP2), requirements management scaling (SP4), scaled sprint planning (SP5), scaled retrospective (SP6), undone department (SP8), and scaled sprint demo/review (SP3, at Kentico, called iteration review). These were the most important practices to allow the adoption of a scaled agile process. Feature teams (SP7) were not among the practices adopted at Kentico.

[‡]Domain-specific language to describe the software behavior without entering the details of implementation, https://github.com/cucumber/cucumber/wiki/Gherkin

- **Scaled retrospective**: The format of the scaled retrospective was the result of several iterations. The company first tried to generate topics during team retrospectives, but there was not enough time and context for identification of major cross-team impediments. Another experiment was to let agile coaches create the topics, but this included only their point of view. The addition of the voting system, together with the topics generated by anyone, further improved the scaled retrospective, as only people interested in the discussed topic attended.
- **Scaled planning**: revolved around so-called Milestones. A Milestone defined vision, goals, business, and technical initiatives the company wanted to achieve in the next release. An iteration of a milestone was 16 weeks long with 4 phases (discovery, strategy, substrategy and go-to-market, Development). The discovery phase identified the most valuable directions of the product, taking into account emerging technologies, market, and competitive analysis. The strategy phase generated and also validated solutions based on the discovery phase. The substrategy subphase was to create an initial roadmap for the new release, identify any obstacles, dependencies, and risks associated with the roadmap, and define steps to mitigate them. The go-to-market used the roadmap from the first subphase, setting-up a document summarizing what the new release is selling, to whom and how. The development phase dealt with the implementation of the previously specified goals in 2 weeks sprint iterations.
- **Scaled sprint demo/review (called iteration review)**: took place biweekly at the end of every development iteration. The primary goal was to spread knowledge across development teams—the meeting was mandatory for everyone in Kentico Cloud. Another goal of the meeting was to collect early feedback and incorporate it early. The meeting also evaluated if everything was delivered or something was missing. Furthermore, the subsequent steps and future of the product were discussed.
- **Scrum of Scrums**: A first meeting was a synchronization meeting for Scrum masters, development director, and other specialists, such as quality assurance coordinator. A second meeting was called Kentico Cloud Sync, and it was a synchronization meeting for the whole Cloud platform. It was attended by agile coaches, development director, and development teams. The current format was the result of multiple iterations of improvements, for example, knowledge sharing was separated to a different meeting which took place before SoS meetings.
- **Requirements management scaling**: Management of cross-team requirements did not follow any particular process. Teams were communicating and synchronizing on items, which had to be implemented by multiple teams, using ad hoc meetings and visiting other teams on a daily basis. The teams could also have some team meetings, such as groomings, together with the other teams. Furthermore, team members could be exchanged between teams, and a cross-team review process was established. The same ad hoc process was used by product managers. All features were initially prioritized in one backlog. After the initial prioritization, each feature was handled in the backlog of the team that was responsible of implementation. If more teams were cooperating on a single feature, they used the backlog of the team that owned the feature.
- **Undone department**: The company used many different specialized teams that helped the development teams with their work. There were an architecture team, a DevOps team, a security team, a team of technical writers, and one user experience team. Each team had one team member who was also a member of one of these expert teams.
- **Communities of practice**: The company used a similar concept called Guilds. Topics of these Guilds were, for example, performance improvements, graphical design, coaching, continuous integration and automatic testing. The number of people involved in these Guilds ranged from 5 to 10. Guilds met at average every 2 weeks.

5.4.3 | Challenges

During the transformation process, we identified 4 challenges that the case company faced. They were resistance to change (SC1), quality assurance issues (SC3), integrating with nonagile parts of the organization (SC4), and too fast roll-out (SC not identified before).

- **Resistance to change**: Although many people in the company were already very experienced with agile development, there were still some that did not like the new way of working. The problem with resistance was also caused by the fact that the development teams—which were given absolute freedom and chose their path—were forced to change. The teams did not want to abandon their previously created process and selected tools.
- **Too fast roll-out**: There were different situations where problems emerged because of the rushed adoption of a new process. Utilizing piloting or smaller gradual changes with 1 or 2 teams could have saved a lot of company's resources.
- **Quality assurance issues**: At the beginning of company's new Cloud platform, the development teams were given absolute freedom. The teams were developing separate products, which were still at the phase of validation. The company let the teams create their entire development process and select their tools and technologies because there was no need to unify the processes, as the development teams almost did not interact among themselves. Although this created a great atmosphere in teams and improved their motivation, the company soon realized that the development teams did not manage their processes properly. Technical debt piled up, and development infrastructure was not set-up properly.
- **Integrating the previous and nonagile parts of organization**: Many parts of the organization were still adopting a nonagile development process, as the new agile development process was created next to the old development model of a previous product. People were moved from the old development process to the new one as necessary. However, the former product still needed maintenance. Some resources, such as technical writers, needed to be shared between the 2 processes. At the end of our research, the company tried to resolve this by including the shared resources into each team. For example, each team would have one technical writer.

5.4.4 | Success factors

Overall, we identified 4 main success factors that helped the company with the adoption of agile development. They were unification of views and values (SF2), executive sponsorship/management support (SF7), company culture (new), and prior agile and lean experience (new).

- **Company culture and prior agile and lean experience**: Many people in the company had prior experience in agile and lean development, especially agile coaches and management. Therefore, the company did not suffer from lack of knowledge or experience as many other companies. The knowledge was spread by agile coaches which used lean practices. They coached people by listening and asking questions, not forcing them to change. Many individuals in the company were interested in agile and lean in general. They were involved in the agile community and followed latest trends in agile development. The company used exchange programs of agile coaches with other organizations and was also collaborating with universities. Furthermore, some employees presented the company results on university lectures. The culture of the company helped as well. People did not feel observed because of transparency as the environment was very open-minded and relaxed.
- **Executive sponsorship/management support**: The management initiated the change: This helped consistently with the adoption. All required resources were available. Also, managers did not try to suffocate employees but helped them grow. The management was also actively involved with the teams during the test phase of new processes. Some members of management, such as CEO and CTO, were members of the development team that tried out the new process. This engagement of executives improved motivation of other team members, as they saw the interest from the management. It also helped the management, because they better understood the practices and the processes.
- **Unification of view and values**: The company defined shared values: It was achieved by introducing an "Agile and Lean Manifesto." The Manifesto contained 12 statements about the core values and principles the company wanted to follow. Additionally, it contained several rules that guided the way of working in the company. These rules defined the priorities in the organization, for example, (1) *cross-team cooperation and consistency before autonomy* and (2) *specialization with substitutability before team responsibility distribution*. The Manifesto helped the company to enforce the adoption of the values.

5.4.5 | Suggestions from the action research

During the action research process, several suggestions for the scaling process were put forward but could not be evaluated given the time frame of the research. They are suggestions that were given the company but will be part of future release cycles, if considered relevant.

The first suggestion is to "promote better engineering practices and debt management." We saw that technical debt and bad engineering practices caused a lot of challenges in large-scale agile.[73,75,76,79,80] Therefore, having solid engineering practices and debt management are some of the prerequisites for successful large-scale agile development. A robust development infrastructure with proper continuous integration and automated tests allows teams to focus more on the new way of working. Teams will also have more time to think about improvements to their development process. If technical debt and bad quality code will pressure the development teams too much, they will not be able to focus on other responsibilities that agile development brings. Teams might also experience overall discouragement: they might start to skip bugs fixing or report a false status of their work. Long and Starr[75] improved their debt management by introducing visual indications of the state of their products, to decide if the product was ready for a new feature or defects had to be solved first. A similar solution is described in detail in dos Santos et al.[96] The paper also adds further motivation of teams for their technical debt management by introducing elements of gamification.[96] Additionally, Long and Starr,[75] Schnitter and Mackert,[76] and Paasivaara et al[80] suggested additional training, use of cross-team backlog, and DevOps teams. However, this problem may not be easily resolved: It may take years.[73]

A second suggestion is to be "careful about too much specialization without knowledge sharing." Communication plays a key role within and across agile development teams.[97] The case company was using specialization in the development teams. It also provided good arguments about the reasons. However, we saw that too much specialization in teams might considerably damage teamwork[73] and create a highly competitive environment.[76] It reduced the overall productivity of teams and caused several other issues. Teams did not have clear, unified view of the development and team plan, as specialized team members focused more on their particular area. The team members only did their particular job. They did not understand the work of other team members and hence distributed decision making was not possible. Moreover, individual team members became unsubstitutable. Therefore, they did not want to share their knowledge and know-how with their colleagues.[73] We suggested considering establishing a process for knowledge sharing between team members early, for example, using workshops or establishing CoPs. In later stages, there is the risk that team members might consider their knowledge too important to share it.

The third suggestion is to "use feature teams and divide the product into customer-centered development areas." Both LeSS and SaFE promote feature teams.[27,35,36] One of the characteristics of feature teams is that they are cross-component. The reason for establishing cross-component teams is that they allow much easier scaling of agile development, as feature teams minimize dependencies between teams. They also enable a much more flexible structure of the organization, because the teams can work on any component and they are not constrained by technical knowledge. Hence, teams can be easily moved between different development areas. As there are no technical constraints, the teams can be divided into development areas, which are centered around the customer: This allows teams to see the whole features from a customer point of view and not only part of the features under implementation in their components. Implementing feature teams is not easy.[79,80] LeSS suggested way to transition to feature teams is to establish one feature team from individuals with knowledge about each component.[27] Team members of such feature teams share their knowledge and work together on a feature that spans between multiple components.[27] The organization should not add more feature

teams until the first is performing well.[27] However, there is a risk involved in this strategy, as a similar one was reported in Paasivaara et al,[80] and it caused problems as team members of the first feature team had central roles in their previous teams.

The fourth suggestion is to "support and promote communities of practice": CoPs are used in both SAFe and LeSS.[27,35,36] They are an instrument that allows voluntary cooperation in an arbitrary area. The most common use of CoPs is knowledge sharing. Thus, they can be utilized to enable better transition to feature teams, to promote better engineering practices or to avoid too much specialization in teams. CoPs are self-organized and are not part of a process or organization entity. However, to establish working CoPs, an organization must actively support and promote them by allocating resources: This means that an organization should provide facilitators, the IT infrastructure and budget. CoPs also need a capable leader, that coordinates and motivates the community. Therefore, organizations should try to find, motivate, and support CoP leaders. There are many ways to use CoPs. Paasivaara and Lassenius[84] studied the use of CoPs with the goals of coaching, improving software craftsmanship, researching emerging technologies, uplifting work and improving coordination and design across teams.[84]

A final suggestion is to better "deploy indicators and metrics to measure progress of the development process": research of existing adoptions of agile showed that the adoption is a continuous process with a long-term goal.[9] This transformation takes effort, resources, and time. Therefore, there is a need for better understanding and evaluation. This can be achieved by quantitatively measuring the change. The transformation also needs to be gradually tailored to the particular context of the organization. Therefore, metrics can be used as a valid argument for this process adjustment. Many organizations struggled to find meaningful metrics to measure their agile transformation.[77,80,83] The reason is that in a traditional waterfall development process, there are clear inputs and outputs of process resources streams. On the other hand, agile development has much more an ad hoc nature: streams of process resources might change daily. Therefore, it is very challenging to find relevant metrics. However, there are metrics that can be used to measure the agile transformation process in different areas, such as change in responsiveness, change in throughput, change in workflow distribution and change in quality.[98] Having such metrics and indicators in place can be useful to track the agile scaling process.

5.5 | Threats to validity

The action research methodology was deemed appropriate for the goals of the current research, as authors wanted to assist the company with the scaling the agile processes and at the same time researching on the way the adoption was performed.[38,99] It also gave the opportunity of benefits beyond other research methodologies, as it has been often reported: "in action research, the emphasis is more on what practitioners do than on what they say they do."[47]

The action research methodology sometimes faces philosophical issues about its scientific correctness as the method is empirical, experimental, observational, and yet it is interpretative, multivariate, and interventionist.[46] It is a research methodology that requires vast amount of time as there needs to be enough time to impact on the processes and to observe the results. This paper involved 8 months of direct contact with the company, with some parts of the transformation process still ongoing. Years might be required to observe the full effects of some changes. Although the application of action research in empirical Software Engineering is limited,[100] we believe that such research method can provide more benefits to industry than traditional case study research[38,101]—being the final goal of Software Engineering research to support practical software development with relevant contributions.[100] Action research can be fitting in this context, as while qualitative research is research *about* practice, action research is research *with* practitioners.[93]

We are aware that the conducted action research is limited because of the possibility of researchers to impact in the development process. In our case, one limitation is that we conducted most of the process through a main contact point at the company. However, we created a research protocol that would allow fast feedback loops from researchers to the development teams in the application of the techniques. Such protocol was based on the application of dialogical action research.[91] This is in line with the fact that in action research, the level of participation of researchers can be be placed on a different spectrum. Learning from the action process is what matters,[93] with the ultimate goal of bringing together *"action and reflection, theory and practice"*.[37] To reduce these threats, we applied the framework from Lau[102] to enhance action research quality. In particular, we rigorously ensured feedback cycles for the actions performed. However, being our approach nearer to action science,[94] we could not cover all aspects.

Another threat to validity is confirmation bias,[103] in that we could have interpreted evidence collected according to our expectations derived from the main literature review. We believe that this threat is limited in the current work, as we did not set a-priori hypotheses to falsify,[104] rather reported findings from the field by different stakeholders. Still, our approach might have lead towards getting answers adjusted according to our previous beliefs.

About construct validity of the action research, the main threat is about how much the organization considered complies with the definition of "large-scale". We believe the organization is representative of such concept, taking into account the two main definitions of large-scale presented in the introduction.[34,48] In our case, we considered four main teams of 7+ members, plus additionally three other cross-cutting teams (architecture team, user experience team and team of technical writers). Considering all the other roles presented (security specialists, quality coordinators, etc …) the amount sums up to more than seven teams and more than 50 members involved.

We cannot *generalize* the results to other cases, as the case reported one single instance and other replications would be necessary. The challenge is on finding other large organizations in the process of scaling their development processes. Nevertheless, given the complexity of organizational contexts, we would expect slightly different results, as the literature review in the first part of the paper has shown. Even the most successful practices might fail in a different context, as the importance of the context is well known in Software Engineering.[105]

As reported, we conducted two loops of the action research cycle in our research. We were limited by the length of software development process adjustments and by time constraints. Overall, the results can be considered gathered over a period of eight months of continuous collaboration with the company. Even though some transformation processes were still ongoing, we consider the time devoted to the research adequate to collect answers to the research questions.

6 | SUMMARY OF THE FINDINGS

Overall, we identified many practices, challenges, and success factors for scaling agile in large organizations (Figure 8, in which we compare findings from the literature and the action research part). We saw that a successful scaling of agile in a large company does not need to follow a particular scheme. Quite the contrary, the case company tailored the development process to the needs while also keeping the core values and principles of agile development. The transformation was mainly successful thanks to the agile mindset and prior experience with agile development of the individuals that were in charge of the the transformation process. Diffusing a new mindset by utilizing coaching with lean principles allowed to spread the new values to other team members. Previous experience from other cases of scaling agile can serve as a useful reference to evaluate several alternatives—the outcomes seem, however, very specific to the context. The factors we identified were generally found as positive for the scalability of agile processes, but it is relatively easy to find cases in which an enabling practice is not successful in other contexts. Nevertheless, we believe that the factors we derived from both the literature review and the action research process, together with the experiences we summarized, can be valuable to other organizations in need to scale their software development processes.

We identified challenges and success factors of "large-scale" agile adoption in an action research within a company. We found out that the case company experienced similar problems, challenges, and success factors to the ones we identified in literature. The biggest challenges the company faced were resistance to change (SC1), quality assurance issues (SC3), integrating the previous and nonagile parts of organization (SC4), and too fast roll-out (new).

There were 4 main success factors that were the most beneficial for the adoption in the case company: unification of view and values (SF2), executive sponsorship/management support (SF7), company culture (new) and prior Agile and Lean experience (new).

The main practices adopted were SoS (SP1), Communities of Practice (SP2), Requirements Management Scaling (SP4), Scaled Sprint Planning (SP5), Scaled Retrospective (SP6), Undone Department (SP8), and Scaled Sprint Demo/Review (SP3, at Kentico called Iteration Review). In Table 4, we provide in more detailed form the main findings related to SP1, SP4, SP5, and SP6 and their comparison with findings from previous literature.

6.1 | Findings in related literature

There are 2 main studies that reviewed primary studies on large-scale agile, namely, Version One Report[57] and Dikert et al.[34] The Version One report[57] identified 14 barriers to further agile adoption and gave 5 recommendations for success with scaling agile.

The 5 biggest barriers to agile adoption mentioned in Version One report[57] were *the ability to change the organizational culture* (55%), *general organizational resistance to change* (42%), *preexisting rigid/waterfall framework* (40%), *not enough personnel with the necessary agile experience* (39%), and *management support* (38%).

The ability to change the organizational structure and the barrier *general organizational resistance to change* are both identified in our research as the challenge *resistance to change*. The barrier *preexisting rigid and waterfall framework* was not found in our research as a barrier. The barrier *not enough personnel with the necessary agile experience* was considered in our research as a challenge; it was included in our challenge *Lack of knowledge, coaching, and training*. The last barrier *management support*, which was considered by the Version One report[57] as *not enough management support*, was not identified in our research. However, *executive sponsorship* was one of the success factors in our study.

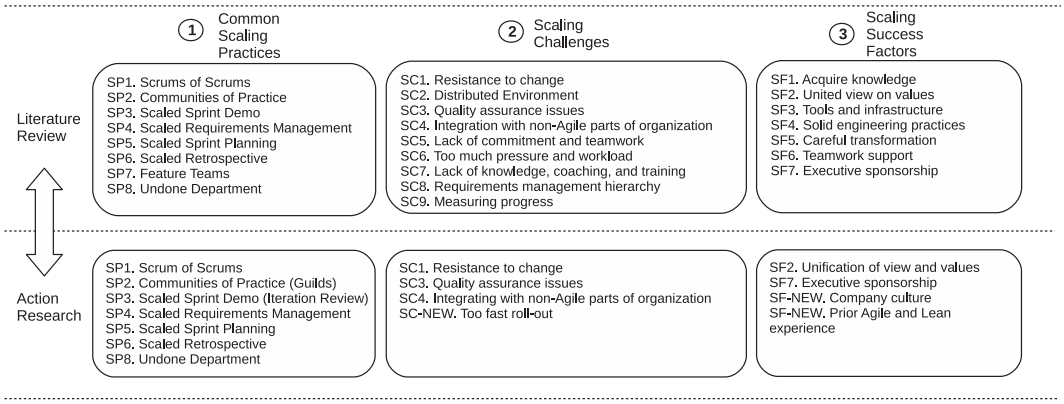

FIGURE 8 Overall research findings (RQ1, RQ2, RQ3, RQ4, RQ5, and RQ6)

TABLE 4 Comparison of main scaling practices at Kentico (SP1, SP4, SP5, and SP6) compare with literature

Scaling Practices	Action Research (Kentico)	Literature
Scrum of Scrums (SP1)	- Several tiers of SoS. - Every topic is time boxed, when the time box expires the whole audience agrees on proceeding further. - Two SoS meetings per week were too many. - Sharing knowledge about topics during meetings was inefficient: separation of knowledge sharing to different meetings. - The meeting needs to be properly prepared (topics need to be described, no new ad-hoc topics added during the meeting. attendance has to understand the context of the topics).	- SoS takes too long when projects grow. Teams were told to report only impediments, this can lead to decrease of collaboration of teams as they might feel nothing is important enough to be reported.[74] - One solution can be to use representatives of teams when meetings grow too large–important is to rotate people in common meetings. One problem involving Scrum Masters is that they are not directly involved with team work and do not have sufficient contextual information.[74] - Daily SoS reduced to weekly.[79] - Separate feature specific SoS from general ones, and create a hierarchy of SoS.[79] SoS can be used for synchronization.[72]
Scaled Requirements Management (SP4)	- One backlog for the prioritization: after prioritization is over the items are further handled in the team backlog. - It may happen that more teams are cooperating over one backlog if they are working together on particular features. - No hierarchy is defined, PMs are cooperating (DoP is continuously being informed) - Great for synchronization	- Scaling might include creating a hierarchy of POs[74] or scaling PO role to a team PO.[76] - The main challenge in creating hierarchical backlog management is to divide the requirements in product areas. Cross-area dependencies can cause huge communication overhead and impossibility to have separate scaled meetings for the product areas.[74] - One idea is to introduce new levels to manage requirements for larger products, e.g. Product Team, Area Product Team. - Clarifying responsibilities, setting up a Product Owner Team can be important.[80]
Common Planning (SP5)	- Different perception of priorities across separate teams. - Helps to focus on the top priority issues in need to be addressed.	- Scaled Sprint Planning might be challenging in a distributed project with multiple sites.[72] - Planning might need more attention and might span across several meetings in large-scale agile development. Additional meetings might be required to explain requirements to teams.[74]
Common Retrospective (SP6)	- Helps to address the issues cross teams/departments. - Improves the way of working across the teams. - Inefficiencies in involving tooo many people. - Many topics are hard to addrses during team retrospectives. - Problems which can occur in one team will be heard by members of othere teams. All get the same context of the issues. - Extrovert people can overrun introverts because of the large group audience. - There might be small "easy-to-fix" topics with high value but the voting mechanism eliminates them.	- Problems which are identified are often too big and hard to solve-might need several iterations to be resolved and the teams might get demoralized when seeing that the problems identified in previous retrospectives are still not resolved and might stop going to the retrospectives. One proposed solution is to set one goal and keep the goal through several iterations until it is resolved.[74]

There are suggestions for a successful agile adoption that Version One report[57] discussed: *consistent process and practices* (43%), *implementation of a common tool across teams* (40%), *agile consultants or trainers* (40%), *executive sponsorship* (37%), and *internal agile support team* (35%).

The suggestion *consistent process and practices* can be considered part of our success factor *careful transformation*, as we mentioned that processes should not be changed very often. *Implementation of common tools across teams* can be regarded as our success factor *tools and infrastructure*. In our research, we identified *knowledge acquisition* as the most important success factor: We mentioned that using agile consultants and trainers was one of the most suggested ways to acquire knowledge. Thus, both suggestions *agile consultants or trainers* and *internal agile support team* can be regarded as this success factor. *Executive sponsorship* was exactly one of our identified success factors.

The other review by Dikert et al[34] examined 52 publications. The paper was published in 2016 and considered only studies after the year 2000. Therefore, the findings are relatively recent. The research found that the most used agile method was Scrum, XP, and Lean software development.[34]

The research by Dikert et al[34] pinpointed 29 success factors in total. The success stories were divided into 9 categories. The most frequently identified categories were *choosing and customizing the agile approach* (48%), *mindset and alignment* (40%), *management support* (38%), *training and coaching* (36%), and *piloting* (36%).

The first success factor *choosing and customizing the agile approach* was not identified by our research. All studied cases we examined customized their development process, but, as we saw, many of them failed. The success factor *mindset and alignment* could be considered as part of our success factors *acquire knowledge* and *common view on values and practices*. Success factors *management support* and *training and coaching* were both identified in our research—we called them *executive sponsorship* and *acquire knowledge*, respectively. The last success factor *piloting* can be considered as our success factor *careful transformation*.

In total, the research by Dikert et al[34] identified 35 challenges, which the authors divided into 9 categories. The biggest challenges of large-scale agile transformation were *agile difficult to implement* (48%), *integrating non-development functions* (43%), *change resistance* (38%), *requirements engineering challenges* (38%), and *hierarchical management and organizational boundaries* (33%).

The first challenge *agile difficult to implement* was not identified in our research. However, the challenge can be caused by lack of knowledge, coaching, and training, which was considered as a challenge in our study. The challenge *integrating non-development functions* can be considered as our challenge *integration with nonagile part of the organization* as many of the nondevelopment parts of the company did not use any agile method. The challenges *change resistance*, and *requirements engineering challenges* were both identified in our research—we called them *resistance to change* and *requirements management hierarchy*, respectively.

Neither of these studies identified *measuring progress* as a challenge. Also, *use of common tools* was considered as a success factor only by the Version One study.[57] Moreover, none of these studies investigated individual scaling practices.

7 | CONCLUSION

The goal of this paper was to collect more evidence about impact factors for scaling agile software development processes inside companies, to better understand practices that work best in a given context.[1,34] By means of a review of studies about large-scale agile in organizations, we identified practices, challenges, and success factors. The initial literature review was the input to conduct an action research study in a software company that was in the process of scaling the agile software development processes. We studied the concrete scaling practices the company utilized, which success factors the company experienced and which challenges the company faced. Thus, we contributed to the existing set of studies about scaling agile with the experience derived from the action research.

Agile culture within the company and prior agile and lean experience, management support, and unification of views and values were found to be key success factors during the action research process. Resistance to change, too quick roll-out, quality assurance issues, and the integration with previous nonagile parts of the organization were found to be critical challenges in the scaling process. Overall, a positive outcome from the action research process was to cross-fertilize ideas from literature to the practical context of the company.

We mentioned several challenges that are connected to large-scale agile. These challenges need more detailed investigation in cooperation with large companies in order to find appropriate solutions. More studies are necessary to find causal relationships between factors.

ORCID


Bruno Rossi http://orcid.org/0000-0002-8659-1520

REFERENCES

1. Eklund U, Berger C. Scaling agile development in mechatronic organizations: a comparative case study. In: Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track. IEEE Press; 2017; Buenos Aires, Argentina:173-182.
2. Fitzgerald B, Stol K-J, O'Sullivan R, O'Brien D. Scaling agile methods to regulated environments: an industry case study. In: 2013 35th International Conference on Software Engineering (ICSE). IEEE; 2013; San Francisco, CA, USA:863-872.
3. Boehm B, Turner R. Management challenges to implementing agile processes in traditional development organizations. *IEEE Software*. 2005;22:30-39.
4. Rolland KH. Scaling across knowledge boundaries: a case study of a large-scale agile software development project. In: Proceedings of the Scientific Workshop Proceedings of XP2016, Vol. 5. ACM; 2016; Edinburgh, Scotland.
5. Carlile PR. A pragmatic view of knowledge and boundaries: boundary objects in new product development. *Organ Sci*. 2002;13:442-455.
6. Carlile PR. Transferring, translating, and transforming: an integrative framework for managing knowledge across boundaries. *Organ Sci*. 2004;15:555-568.
7. Dyba T, Dingsoyr T. What do we know about agile software development?. *IEEE Software*. 2009;26:6-9.
8. Ambler SW. Scaling agile software development through lean governance. In: ICSE Workshop on Software Development Governance, 2009. SDG'09. IEEE; 2009; Washington, DC, USA:1-2.
9. Ebert C, Paasivaara M. Scaling agile. *IEEE Software*. 2017;34:98-103.

10. Paasivaara M. Adopting SAFe to scale agile in a globally distributed organization. In: 2017 IEEE 12th International Conference on Global Software Engineering (ICGSE). IEEE; 2017:36-40.
11. Kasauli R, Liebel G, Knauss E, Gopakumar S, Kanagwa B. Requirements engineering challenges in large-scale agile system development. In: Requirements Engineering Conference (RE), 2017 IEEE 25th International. IEEE; 2017; Lisbon, Portugal:352-361.
12. Mishra D, Mishra A. Complex software project development: agile methods adoption. *J Softw Maint Evol Res Pract*. 2011;23:549-564.
13. Heikkila V, Rautiainen K, Jansen S. A revelatory case study on scaling agile release planning. In: 2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA). IEEE; 2010; Lille, France:289-296.
14. Power K. A model for understanding when scaling agile is appropriate in large organizations. In: International Conference on Agile Software Development. Springer; 2014:83-92.
15. Saddington P. Scaling agile product ownership through team alignment and optimization: a story of epic proportions. In: Agile Conference (AGILE), 2012. IEEE; 2012:123-130.
16. Moore E, Spens J. Scaling agile: finding your agile tribe. In: Agile 2008 Conference; 2008; Toronto:121-124.
17. Conboy K, Coyle S, Wang X, Pikkarainen M. People over process: key challenges in agile development. *IEEE Software*. 2011;28:48-57.
18. Dingsøyr T, Moe Nils B, Fægri TE, Seim EA. Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation. *Empir Softw Eng*. 2017;23:1-31.
19. Paasivaara M, Lassenius C. Challenges and success factors for large-scale agile transformations: a research proposal and a pilot study. In: Proceedings of the Scientific Workshop Proceedings of XP2016, Vol. 9. ACM; 2016; Edinburgh, Scotland, UK.
20. Chow T, Cao D-B. A survey study of critical success factors in agile software projects. *J Syst Softw*. 2008;81:961-971.
21. Rossi B, Russo B, Succi G. Open source software and open data standards as a form of technology adoption: a case study. In: IFIP International Conference on Open Source Systems. Springer; 2007:325-330.
22. Rossi B, Russo B, Succi G. Adoption of free/libre open source software in public organizations: factors of impact. *Inf Technol People*. 2012;25:156-187.
23. Rossi B, Russo B, Succi G. Evaluation of a migration to Open Source Software. *Handbook of Research on Open Source Software: Technological, Economic*; 2007:309.
24. Fitzgerald B. Open source software adoption: anatomy of success and failure. *Multi-Disciplinary Advancement in Open Source Software and Processes*. 2011:1-23.
25. Fitzgerald B, Kesan JP, Russo B, Shaikh M, Succi G. *Adopting Open Source Software: A Practical Guide*. Cambridge, MA: The MIT Press; 2011;186.
26. Wang X, Conboy K, Pikkarainen M. Assimilation of agile practices in use. *Inf Syst J*. 2012;22:435-455.
27. Larman C, Vodde B. *Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum*. New York: Pearson Education; 2010.
28. Larman C, Vodde B. Scaling agile development. *CrossTalk*. 2013;9:8-12.
29. Reifer DJ, Maurer F, Erdogmus H. Scaling agile methods. *IEEE Software*. 2003;20:12-14.
30. Freudenberg S, Sharp H. The top 10 burning research questions from practitioners. *Ieee Software*. 2010;27:8-9.
31. Paasivaara M, Lassenius C. Scaling scrum in a large distributed project. In: 2011 International Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE; 2011:363-367.
32. Kasauli R, Knauss E, Nilsson A, Klug S. Adding value every sprint: a case study on large-scale continuous requirements engineering. In: REFSQ Workshops; 2017; Essen, Germany.
33. Hobbs B, Petit Y. Agile methods on large projects in large organizations. *Proj Manag J*. 2017;48:3-19.
34. Dikert K, Paasivaara M, Lassenius C. Challenges and success factors for large-scale agile transformations: a systematic literature review. *J Syst Softw*. 2016;119:87-108.
35. Scaled Agile Inc.. SAFe 4.0 Introduction A Scaled Agile Inc.. White Paper July 2016 Overview of the Scaled Agile Framework for Lean Software and Systems Engineering. tech. rep., Scaled Agile, Inc.5480 Valmont Rd, Suite 100, Boulder CO 80301 USA; 2016.
36. Leffingwell D. *SAFe® 4.0 Reference Guide: Scaled Agile Framework® for Lean Software and Systems Engineering*: Addison-Wesley Professional; 2016.
37. Brydon-Miller M, Greenwood D, Maguire P. Why action research?; 2003.
38. Easterbrook S, Singer J, Storey M-A, Damian D. Selecting empirical methods for software engineering research. *Guide to Advanced Empirical Software Engineering*. 2008:285-311.
39. Schön E-M, Thomaschewski J, Escalona MJ. Agile requirements engineering: a systematic literature review. *Computer Standards & Interfaces*. 2017;49:79-91.
40. Torrecilla-Salinas CJ, Sedeño J, Escalona MJ, Mejías M. Agile, Web engineering and capability maturity model integration: a systematic literature review. *Inf Softw Technol*. 2016;71:92-107.
41. Budgen D, Turner M, Brereton P, Kitchenham B. Using mapping studies in software engineering. In: Proceedings of PPIG. Lancaster University; 2008; United Kingdom:195-204.
42. Petersen K, Feldt R, Mujtaba S, Mattsson M. Systematic mapping studies in software engineering. In: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, EASE'08. BCS Learning & Development Ltd.; 2008; Swindon, UK:68-77.
43. Pergher M, Rossi B. Requirements prioritization in software engineering: a systematic mapping study. In: 2013 IEEE Third International Workshop on Empirical Requirements Engineering (EmpiRE). IEEE; 2013:40-44.
44. Sjoberg DIK, Dyba T, Jorgensen M. The future of empirical methods in software engineering research. In: 2007 Future of Software Engineering. IEEE Computer Society; 2007; Los Alamitos, CA:358-378.
45. Coughlan P, Coghlan D. Action research for operations management. *Int J Oper Prod Manag*. 2002;22:220-240.
46. Baskerville RL, Wood-Harper AT. *A Critical Perspective on Action Research as a Method for Information Systems Research*. Cham: Springer International Publishing; 2016. 169–190.

47. Avison DE, Lau F, Myers MD, Nielsen PA. Action research. *Commun ACM*. 1999;42:94-97.
48. Dingsøyr T, Fægri TE, Itkonen J. *What Is Large in Large-Scale? A Taxonomy of Scale for Agile Software Development*. Cham: Springer International Publishing; 2014. 273–276.
49. Dingsøyr T, Moe NB. Towards principles of large-scale agile development. In: International Conference on Agile Software Development. Springer; 2014:1-8.
50. Moe NB, Dingsøyr T. Emerging research themes and updated research agenda for large-scale agile development: a summary of the 5th international workshop at XP2017. In: Proceedings of the XP2017 Scientific Workshops, Vol. 14. ACM; 2017.
51. Alqudah M, Razali R. A review of scaling agile methods in large software development. *Int J Adv Sci Eng Inf Technol*. 2016;6:828-837.
52. Sutherland J. Agile can scale: inventing and reinventing SCRUM in five companies. *Cutter IT Journal*. 2001;14(12):5-11.
53. Ambler SW, Lines M. *Disciplined agile delivery: A practitioner's guide to agile software delivery in the enterprise*. Upper Saddle River, NJ: IBM Press; 2012.
54. Schwaber K, Nexus Guide. The definitive guide to Nexus: the exoskeleton of scaled Scrum development. *PDF). scrum. org*. 2015.
55. Thompson K. Recipes for agile governance in the enterprise; 2013.
56. One V. 11th annual state of agile survey. Technical report, Version One; 2017.
57. One V. 10th annual state of agile survey. Technical report, Version One; 2016.
58. Alliance A. Scrum of Scrums. *Online at http://www. agilemanifesto. org*. 2001;6.
59. Frank A, Hartel C. Feature teams collaboratively building products from ready to done. In: Agile Conference, 2009. AGILE'09. IEEE; 2009:320-325.
60. Ambler SW, Lines M. Going beyond Scrum: disciplined agile delivery. *Disciplined Agile Consortium. White Paper Series*. 2013:1-16.
61. Bittner K, Kong P, Naiburg E, West D. *The Nexus Framework for Scaling Scrum: Continuously Delivering an Integrated Product with Multiple Scrum Teams*: Addison-Wesley Professional; 2017.
62. Wenger E. *Communities of Practice: Learning, Meaning, and Identity*. New York: Cambridge University Press; 1998.
63. Kahkonen T. Agile methods for large organizations-building communities of practice. In: Agile Development Conference, 2004. IEEE; 2004; Los Alamitos, CA:2-10.
64. Borzillo S, Schmitt A, Antino M. Communities of practice: keeping the company agile. *J Bus Strateg*. 2012;33:22-30.
65. Kitchenham BA, Dyba T, Jorgensen Magne. Evidence-based software engineering. In: Proceedings of the 26th International Conference on Software Engineering. IEEE Computer Society; 2004; Edinburgh, Scotland:273-281.
66. Dyba T, Kitchenham BA, Jorgensen M. Evidence-based software engineering for practitioners. *IEEE software*. 2005;22:58-65.
67. Kitchenham B, Brereton OP, Budgen D, Turner M, Bailey J, Linkman S. Systematic literature reviews in software engineering—a systematic literature review. *Inf Softw Technol*. 2009;51:7-15.
68. Arksey H, O'Malley L. Scoping studies: towards a methodological framework. *Int J Soc Res Methodol*. 2005;8:19-32.
69. Budgen D, Brereton P. Performing systematic literature reviews in software engineering. In: Proceedings of the 28th International Conference on Software Engineering. ACM; 2006; Shanghai, China:1051-1052.
70. Benzies KM, Premji S, Hayden KA, Serrett K. State-of-the-evidence reviews: advantages and challenges of including grey literature. *Worldviews Evid-Based Nurs*. 2006;3:55-61.
71. Hayes W. Research synthesis in software engineering: a case for meta-analysis. In: Software Metrics Symposium, 1999. Proceedings. Sixth International. IEEE; 1999:143-151.
72. Vallon R, Strobl S, Bernhart M, Grechenig T. Inter-organizational co-development with scrum: experiences and lessons learned from a distributed corporate development environment. In: International Conference on Agile Software Development. Springer; 2013; Berlin:150-164.
73. Moe NB. *Key Challenges of Improving Agile Teamwork*. Berlin, Heidelberg: Springer Berlin Heidelberg; 2013. 76–90.
74. Paasivaara M, Lassenius C. Scaling Scrum in a large globally distributed organization: a case study. In: 2016 IEEE 11th International Conference on Global Software Engineering (ICGSE); 2016:74-83.
75. Long K, Starr D. Agile supports improved culture and quality for healthwise. In: Proceedings of the Agile 2008, AGILE '08. IEEE Computer Society; 2008; Washington, DC, USA:160-165.
76. Schnitter J, Mackert O. *Large-Scale Agile Software Development at SAP AG*. Berlin, Heidelberg: Springer Berlin Heidelberg; 2011. 209–220.
77. Atlas A. Accidental adoption: the story of Scrum at Amazon.com. In: 2009 Agile Conference; 2009:135-140.
78. Hanly S, Wai L, Meadows L, Leaton R. Agile coaching in British Telecom: making strawberry jam. In: AGILE 2006 (AGILE'06), Vol. 9; 2006:202.
79. Paasivaara M, Lassenius C, Heikkila VT, Dikert K, Engblom C. Integrating Global sites into the lean and agile transformation at Ericsson. In: 2013 IEEE 8th International Conference on Global Software Engineering; 2013:134-143.
80. Paasivaara M, Behm B, Lassenius C, Hallikainen M. Towards rapid releases in large-scale XaaS development at Ericsson: a case study. In: 2014 IEEE 9th International Conference on Global Software Engineering; 2014:16-25.
81. Schatz B, Abdelshafi I. Primavera gets agile: a successful transition to agile development. *IEEE Software*. 2005;22:36-42.
82. Hajjdiab H, Taleb AS, Ali J. An industrial case study for scrum adoption. *J Softw(Oulu)*. 2012;7.
83. Benefield G. Rolling out agile in a large enterprise. In: 2008 41st Annual Hawaii International Conference on System Sciences, Vol. 00; 2008:462.
84. Paasivaara M, Lassenius C. Communities of practice in a large distributed agile software development organization—Case Ericsson. *Inf Softw Technol*. 2014;56:1556-1577. Special issue: Human Factors in Software Development.
85. Mohammadi E, Thelwall M, Haustein S, Larivière V. Who reads research articles? An altmetrics analysis of Mendeley user categories. *J Assoc Inf Sci Technol*. 2015;66:1832-1846.
86. Mishra A, Garbajosa J, Wang X, Bosch J, Abrahamsson P. Future directions in agile research: alignments and divergence between research and practice. *J Softw Evol Proc*. 2017;29.

87. Runeson P. It takes two to Tango—an experience report on industry–academia collaboration. In: 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST). IEEE; 2012:872-877.
88. Wohlin C. Empirical software engineering research with industry: top 10 challenges. In: Proceedings of the 1st International Workshop on Conducting Empirical Studies in Industry. IEEE Press; 2013:43-46.
89. Garousi V, Petersen K, Ozkan B. Challenges and best practices in industry-academia collaborations in software engineering: a systematic literature review. *Inf Softw Technol*. 2016;79:106-127.
90. Martínez-Fernández S, Marques HM. Practical experiences in designing and conducting empirical studies in industry-academia collaboration. In: Proceedings of the 2nd International Workshop on Conducting Empirical Studies in Industry. ACM; 2014:15-20.
91. Mårtensson P, Lee AS. Dialogical action research at omega corporation. *MIS Quarterly*. 2004:507-536.
92. Baskerville R, Wood-Harper AT. Diversity in information systems action research methods. *Eur J Inf Syst*. 1998;7:90-107.
93. Bradbury-Huang H. What is good action research? Why the resurgent interest?. *Action Res*. 2010;8:93-109.
94. Argyris C, Schön DA. Participatory action research and action science compared: a commentary. *Am Behav Sci*. 1989;32:612-623.
95. Sato DT, Corbucci H, Bravo MV. Coding dojo: an environment for learning and sharing agile practices. In: Agile, 2008. AGILE'08. Conference. IEEE; 2008:459-464.
96. Santos P, Sérgio M, Varella A, Dantas CR, Borges DB. Visualizing and managing technical debt in agile development: an experience report. Springer Berlin Heidelberg; 2013; Berlin, Heidelberg:121-134.
97. Pikkarainen M, Haikara J, Salo O, Abrahamsson P, Still J. The impact of agile practices on communication in software development. *Empir Softw Eng*. 2008;13:303-337.
98. Olszewska M, Heidenberg J, Weijola M, Mikkonen K, Porres I. Quantitatively measuring a large-scale agile transformation. *J Syst Softw*. 2016;117:258-273.
99. Davison R, Martinsons MG, Kock N. Principles of canonical action research. *Inf Syst J*. 2004;14:65-86.
100. Santos PSM, Travassos GH. Action research use in software engineering: an initial survey. In: 3rd International Symposium on Empirical Software Engineering and Measurement, 2009. ESEM 2009. IEEE; 2009:414-417.
101. Dos Santos PSM, ravassos GH. Action research can swing the balance in experimental software engineering. In: Advances in Computers. Elsevier; 2011:205-276.
102. Lau F. Toward a framework for action research in information systems studies. *Inf Technol People*. 1999;12:148-176.
103. Nickerson RS. Confirmation bias: a ubiquitous phenomenon in many guises. *Rev Gen Psychol*. 1998;2:175.
104. Mynatt CR, Doherty ME, Tweney RD. Confirmation bias in a simulated research environment: an experimental study of scientific inference. *Q J Exp Psychol*. 1977;29:85-95.
105. Dybå T, Sjøberg DIK, Cruzes DS. What works for whom, where, when, and why? On the role of context in empirical software engineering. In: 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE; 2012:19-28.

How to cite this article: Kalenda M, Hyna P, Rossi B. Scaling agile in large organizations: Practices, challenges, and success factors. *J Softw Evol Proc*. 2018;30:e1954. https://doi.org/10.1002/smr.1954