Empir Software Eng
DOI 10.1007/s10664-017-9524-2

Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation

Torgeir Dingsøyr[1,2] · **Nils Brede Moe**[1] · **Tor Erlend Fægri**[1] · **Eva Amdahl Seim**[3]

Abstract Agile development methods were believed to best suit small, co-located teams, but the success in small teams has inspired use in large and very large-scale software development. However, fundamental assumptions of agile development are challenged when applying the methods at a very large scale. An interpretative revelatory case study on one of the largest software development programmes in Norway shows how agile methods were adapted and complemented with practices from traditional methods to handle the scale. The programme ran over four years with 12 co-located development teams and a total of 175 people involved. The case study was conducted retrospectively using group interviews with 24 participants and documents. Findings on key challenging areas are reported: customer involvement, software architecture, and inter-team coordination. The revelatory study also suggests refinements of a research agenda for very large-scale agile development.

Communicated by:Vittorio Cortellessa

✉ Torgeir Dingsøyr
torgeird@sintef.no

Nils Brede Moe
nils.b.moe@sintef.no

Tor Erlend Fægri
tor.e.faegri@sintef.no

Eva Amdahl Seim
eva.amdahl.seim@sintef.no

1 Department of Software Engineering, Safety and Security, SINTEF, Box 4760 Sluppen, Trondheim 7465, Norway

2 Department of Computer Science, Norwegian University of Science and Technology, Trondheim, Norway

3 Department of Industrial Management, SINTEF, Box 4760 Sluppen, Trondheim, PO, Norway

Keywords Very large-scale software development · Software engineering · Requirements engineering · Project management · Portfolio management · Knowledge management · Customer involvement · Agile software development · Multi team systems · Inter-team coordination · Software architecture · Scrum · Extreme programming

1 Introduction

Since the formulation of the agile manifesto in 2001, agile methods have transformed software development practice by strongly emphasizing change tolerance, evolutionary delivery, and active end-user involvement (Dingsøyr et al. 2012). Rajlich (2006) describes agile development as a paradigm shift in software engineering that "brings a host of new topics into the forefront of software engineering research". In the first special issue on agile development in *IEEE Computer*, Williams and Cockburn (2003) stated that agile methods "best suit *collocated teams* of about *50 people or fewer* who have easy access to user and business experts and are developing projects that are not life-critical". Agile methods have been criticized for lack of focus on architecture, the little scientific support for claims in the agile community, and for being applicable only to small teams (Dybå and Dingsøyr 2008). Boehm argued that projects need to find a "sweet spot" between traditional and agile development based on the level of risk that the project is willing to take (Boehm 2002; Boehm and Turner 2003).

Since then, the success of agile methods for small, co-located teams has inspired use in new domains, and companies increasingly apply agile practices to large-scale projects. Frameworks for managing large agile development projects have started to appear, such as the Scaled Agile Framework[1] and Large Scale Scrum.[2] New challenges arise as agile development techniques are used on large-scale projects. "Agile in the large" was voted "top burning research question" by practitioners at the XP2010 conference (Freudenberg and Sharp 2010). Practitioners ask questions like, "How do you scale up a large project over many months or even years?" (Gregory et al. 2016).

Large-scale projects or programmes are also fraught with challenges in other industries such as infrastructure, construction, water, and energy. Such "megaprojects" typically use traditional project management methods and are often associated with large cost overruns and delays as the considerable complexities are difficult to manage effectively (Flyvbjerg 2014; Giezen 2012). For companies engaged in development programmes, effective portfolio management is critical for managing the exponential growth of interdependencies and mitigating associated risks (Blichfeldt and Eskerod 2008). However, the iterative nature of agile methods introduces new challenges in portfolio management that necessitate different patterns of action (Nerur and Balijepally 2007; Stettina and Horz 2015).

The term "large-scale agile development" has been used to describe agile development in a range of contexts – spanning large teams to large multi-team projects to making use of principles of agile development in a whole organization. We define large-scale agile development as agile development efforts that involve a large number of actors, a large number of systems and interdependencies (Rolland et al. 2016), which have "*more than two teams*" and very large-scale as "*agile development efforts with more than ten teams"* (Dingsøyr et al. 2014)*,* These definitions exclude agile methods applied

[1] See http://www.scaledagileframework.com/
[2] See https://less.works/

in large organizations (Barlow et al. 2011) from “large-scale agile”, which we consider to be a research direction on its own. Many refer to this as “enterprise agile” (Bass 2015; Fitzgerald and Stol 2017).

Fundamental assumptions in agile development are severely challenged when using these practices in large-scale projects. Self-management is a central principle in agile methods, but studies from fields other than software development indicate that self-management can reduce the ability to coordinate across teams effectively (Ingvaldsen and Rolfsen 2012). Also, while the teams need to self-manage, team members need to have an effective knowledge network and collaborate closely with experts outside the team in large-scale agile environments (Moe et al. 2014; Šmite et al. 2017). Having an emerging architecture (Eckstein 2014) could hamper project progress when many teams are working in parallel. Nord et al. (2014) argue that the tools needed to handle architectural decisions in large projects do not exist in agile methods. Many agile teams working towards the same goal requires much coordination and management effort (Petersen and Wohlin 2010). Practices like Scrum of Scrum have been found to be inefficient in large projects (Paasivaara et al. 2012) as is it hard to find the right level of detail in discussions to keep the forum interesting for many participants. Knowledge sharing focuses on tacit knowledge in agile methods, which is challenged when the number of participants in a project or programme increases. Large-scale programmes often have a high number of stakeholders and users of the product, and their needs need to be communicated to a number of developers. This intensifies the challenges with customer involvement.

The workshop on large-scale agile development at the International Conference on Agile Software Development (XP) has worked to define a research agenda where topics with high priority include “organization of large development efforts”, “variability factors in scaling”, “inter-team coordination”, and “release planning and architecture” (Dingsøyr and Moe 2014).

This article presents an interpretative revelatory case study of a very large-scale development programme with extensive use of the agile method Scrum. We raise the following research question: *How can agile methods be adapted in the very large scale, regarding programme organization, customer involvement, software architecture and inter-team coordination?* Arguments for the chosen topic areas are provided in section 3. The main contribution is a thorough description of method adaptation and identification of key characteristics of adaptation, for example by showing how customer involvement was organized to adapt to both a large number of customer representatives and a large number of developers. Based on these findings, we suggest directions for future research in the form of novel research questions that extend the research agenda on large-scale agile development.

In the following, Section 2 provides an introduction to very large-scale development, first outlining differences between traditional and agile development and then providing background on three focus areas: customer involvement, software architecture and inter-team coordination. Section 3 provides a description of how we conducted the interpretative revelatory case study, selection criteria for the case as well as methods for data collection and analysis. Section 4 describes results, first through a thick description of how the large-scale programme was organized, then providing details on customer involvement, software architecture and inter-team coordination. Section 5 contrasts findings with previous work identified in Section 2 and discusses the main limitations of the study. Section 6 concludes by formulating novel research questions based on our study.

2 Very Large-Scale Development

Boehm and Turner (2003) suggested that programmes should find a “sweet spot” combining a mixture of traditional and agile methods. Previous research on the use of development methods suggests that methods need to be adapted to the work context (Fitzgerald et al. 2006). We first describe some of the main differences between traditional and agile development before focusing on three aspects of adaptation that are critical in very large scale development: customer involvement, software architecture, and inter-team coordination.

2.1 Traditional and Agile Development

Nerur et al. (2005) described the fundamental assumption behind traditional methods to be that information systems are fully specifiable and built through meticulous and extensive planning. Agile methods, on the other hand, assume that information systems can be built through continuous design, improvement, and testing based on rapid feedback and change. Learning and adaptation should be embraced (Conboy 2009). Adapting the method to the context will involve balance in a number of areas (Vinekar et al. 2006), as illustrated in Table 1. Very large-scale projects involve great risk and management attention. Boehm and Turner (2003) argued that traditional and agile methods should be balanced when facing risk such as increases in programme size.

The transition from traditional plan-driven development to a more agile method with iterations and development conducted in small teams was the focus of a study by Petersen and Wohlin (2010). They examined the development of three large subsystem components in an Ericsson product involving 117 people and found that many of the issues raised in traditional development were not raised after the transition to agile development. This suggests that agile methods can also work well in large-scale product development. Another study from Ericsson finds that agile principles in large scale contributed to knowledge-sharing, and led to increased project visibility and effectiveness in coordination (Lagerberg et al. 2013).

One of the very few studies on large projects combining traditional and agile methods examined a project to develop a web-based customer booking engine for an American cruise company (Batra et al. 2010). The study describes a $15 million project that lasted for 28 months. The project was distributed and combined Scrum with the

Table 1 Traditional versus agile development (excerpt from (Nerur et al. 2005))

	Traditional development	Agile development
Management style	Command and control	Leadership and collaboration
Knowledge management	Explicit	Tacit
Communication	Formal	Informal
Development model	Life-cycle model (waterfall, spiral or some variation)	The evolutionary-delivery model
Desired organizational form/ structure	Mechanistic (bureaucratic with high formalization), aimed at large organizations	Organic (flexible and participative encouraging cooperative social action), aimed at small and medium-sized organizations
Quality control	Heavy planning and strict control. Late, heavy testing	Continuous control of requirements, design and solutions. Continuous testing

Project Management Body of Knowledge[3] framework. Customers were available but did not work together with developers on a daily basis. The iteration length was two weeks. Some of the challenges identified in the study were due to the size and involvement of a high number of internal business sponsors, users, project managers, analysts, and external developers from the UK and India; Customer involvement. The communications were mainly formal, and formal documents were needed for changes. However, the project was considered a success, and the study describes the balance between traditional and agile methods as essential in achieving both project control and agility.

A model organizing development as chains of Scrum teams (the output of one team is the input of the next) is described in a study of three cases of organizations with 150, 34, and 5 Scrum teams (Vlietland and van Vliet 2015). The study investigated strategy, structure, collaboration, coordination, communication, mind-set, and competence of people. The dependence of teams on the results from other teams introduces a number of challenges. The three cases illustrate challenges that include a lack of coordination in the chain, mismatch of backlog priority between teams, challenges in alignment between teams, and unpredictability in delivering on commitments, this suggests challenges with inter-team coordination and main design decisions as expressed in the software architecture.

2.2 Customer Involvement

Agile methods are people-centric and recognize the value that competent people and their relationships bring to software development (Nerur and Balijepally 2007). A key pillar in any agile method is the close and continual collaboration between clients and developers (Maiden and Jones 2010). Therefore, agile methods are highly dependent on the on-site customer identifying and prioritizing features, providing feedback, and guiding change in the course of the development (Vinekar et al. 2006). In their study on a large-scale agile project, Bjarnason et al. (2012) found that low customer involvement in near-development roles in combination with weak awareness of overall goals may result in an unrealistically large project scope. Further, overscoping can lead to a number of negative effects, including quality issues, delays, and failure to meet customer expectations. A study on large-scale and distributed projects found that understanding requirement dependencies is of paramount importance in such projects (Daneva et al. 2013).

Active participation and constant involvement of the customer in systems development yields greater benefits, but this reliance on the customer can fail if the on-site customer goals are misaligned with the goals of other stakeholders. Having multiple on-site customers in a large-scale project increases the risk of failure because of the challenge of establishing a common understanding among all customer representatives. A fragmented view of the system that each customer may have is likely to have a negative impact on the project (Ramesh et al. 2010). Further, when different stakeholder groups have different priorities, there is a need for open and transparent dialogue and cross stakeholder group communication in large-scale agile projects (Barney and Wohlin 2009).

[3] See http://www.pmi.org/PMBOK-Guide-and-Standards.aspx

2.3 Software Architecture

The software architecture is the fundamental technical organization of a system. How and when to make architectural decisions have been the subject of major debate in the software engineering field (Abrahamsson et al. 2010). In traditional development, the architecture is defined prior to implementation and testing, whereas the architectural design emerges as a result of on-going clarification in 'purist' agile development. Traditionally, software architecture is associated with up-front designs and stable structures that accommodate pre-defined, non-functional requirements. The assumption is that the value of good architectural decisions will surface at a later point in time in forms such as more easily maintainable code and scalability (Faber 2010). Also, a sound architecture is largely invisible and provides an effective structure for subsequent functionality. It is interesting to note that software architecture has meaning and significance for a wide variety of stakeholders. Different stakeholders are also likely to associate different meanings to software architecture. Smolander (2002) suggested that the four metaphors blueprints, literature, language, and decisions capture the meaning of software architecture for different actors involved in software development. This underlines the pervasive role of software architecture.

Several approaches to architecture work have been taken in large agile projects. Some start with the architecture ('big up-front design') and then use agile methods. Others spend the first iteration focusing on architecture. Others again start directly on development and let the architecture emerge. To construct a large software system developed by a number of teams, it is vitally important for the architecture to be agreed upon and communicated without introducing the bureaucracy and overhead associated with traditional methods. In a study on software product companies, Unphon and Dittrich (2010) found that architectural knowledge was transferred by face-to-face communication with chief architects taking the role of a "walking architecture".

Awareness and social protocols are important perspectives on how architecture is communicated. Unphon and Dittrich do not discuss the increased challenges of architectural work at a large scale. In their study of a large-scale agile approach at Ericsson, Petersen and Wohlin (2010) found a need for a high-level architectural design to facilitate planning. Nord et al. (2014) argued that for large-scale agile projects, agility is enabled by architecture, and architecture is enabled by agility. They suggested several tactics for handling architecture in large-scale projects, including making use of a matrix structure and focusing on the production infrastructure.

2.4 Inter-Team Coordination

Coordination can be defined as "the managing of dependencies" (Malone and Crowston 1994), where dependencies can be related to tasks, knowledge, resources, or technology. The central challenge in coordination is identifying the right form or artefacts, arenas, and degree of formalization in large projects with high uncertainty. In small agile projects, the development team coordinates work through frequent informal interaction among themselves and with customers, as in the customer-on-site practice in eXtreme Programming. Scrum has dedicated meetings for planning, review, and retrospectives. Many teams use visual boards, like in Kanban, to show who is working on what and the status of work tasks. Strode et al. (2012) explain coordination at the team level in agile teams and propose a model for coordination strategy and coordination effectiveness.

For large-scale projects, there is less support. Scrum prescribes regular meetings between Scrum teams ("Scrum of Scrums") in order to manage the interfaces between teams. Eckstein shows techniques that are applicable to large projects in order to facilitate planning, status information, integration, and retrospectives in her book with recommendations to practitioners (Eckstein 2004). Some large-scale agile frameworks have been suggested by practitioners (Larman and Vodde 2013; Larman and Vodde 2017; Leffingwell et al. 2017) that describe roles and arenas for inter-team coordination. Visual boards was the primary form of inter-team coordination in a project in Sweden described in an experience report by Kniberg (2011).

There is a small body of studies on inter-team coordination. Vlietland and van Vliet (2015) propose that embedded coordination practices within and between Scrum teams positively impact delivery predictability in large projects. A study of "Scrum of Scrums" (Paasivaara et al. 2012) suggests that this forum did not lead to satisfactory coordination: feature-specific or site-specific fora were better, but coordination at the project level was still a challenge. Researchers working closely with SAP (Scheerer et al. 2014; Scheerer and Kude 2014) have developed models of coordination called "coordination configurations" and are exploring how coordination configuration influences coordination effectiveness. Paasivaara and Lassenius (2014) describe a very large-scale development initiative at Ericsson with 40 teams where four types of communities of practice (Wenger 1998) are used to coordinate teams. A survey on coordination in large-scale software teams found that respondents wished more effective and efficient communication, as well as the importance of good personal relationships for coordination (Begel et al. 2009).

A management science study (Ingvaldsen and Rolfsen 2012) suggests that inter-group coordination is a major challenge when groups are self-managing. Self-management involves giving teams authority to decide how to 1) execute tasks, and 2) monitor and manage their work process (Hackman 1986). Moe et al. (2009) describe challenges to self-management at the team level and highlight challenges at the organizational level, such as shared resources, organizational control, and specialist culture.

3 Method

We have chosen an interpretative embedded revelatory case study (Klein and Myers 1999; Runeson and Höst 2009; Yin 2014) to investigate the research question of how agile methods can be adapted in the very large scale. There are few empirical studies on how agile methods are adapted at the very large scale, and we seek new insights to generate ideas for research agendas on agile approaches in very large-scale development. We focus on one case that was described by practitioners as a particularly successful very large programme with extensive use of agile methods.[4] We selected a case where the whole development programme was co-located in order to remove effects due to distribution of teams. In the following, we refer to the case as the Perform programme, which aimed to develop a new office automation system for the *Norwegian Public Service Pension Fund (the* "*Pension Fund*"*).* This programme ran from 2008 to 2012 and had at most 12 development teams working in parallel. The development project was divided in

[4] The programme is known for presenting their model at national agile development conferences, and is a programme that was completed on time and budget.

three parts: one carried out by an internal development unit (6 teams) and two other parts conducted by consulting companies *Accenture* and *Steria.*[5] We provide more details of the project in the results section. After an initial meeting with the manager of the development project from the *Pension Fund*, we met with project managers from *Accenture* and *Steria* and were granted access to collect data from all three parties.

We chose an embedded study to gain further insight in topics described in section 2 which are challenging in very large-scale programmes:

1. Customer involvement, because a very-large programme involves a number of requirements, stakeholders and developers.
2. Software architecture, because a very-large programme involves many developers and it is a challenge to adjust main design decisions given many stakeholders.
3. Inter-team coordination, as very-large programmes involve a number of teams who handle tasks with interdependencies and thus need methods that facilitate coordination.

Before starting data collection, we studied public presentations and an official report from the programme in order to increase our understanding. We quickly realized that the programme had a very complex organization with 175 people involved and a number of projects and subprojects. Our data collection started after the project was finished, and all programme personnel were then allocated to new projects. This created challenges in gaining access to people and in that it had been over a year since some had worked on the project.

3.1 Data Collection

We chose to use two types of data for this revelatory study: Group interviews and documents: First, for each topic in the embedded case study design (customer involvement, software architecture, inter-team coordination), we organized three group interviews (Myers and Newman 2007), one for each organization. The reason for three interviews on each topic was that we knew from introductory meetings that there would be slightly different approaches in the three development subprojects, and having meetings in each organization might have led to more openness (and trust as discussed by Myers and Newman (2007)) than in joint group meetings. Due to the explorative character of the study we primarily wanted groups in order to facilitate a broader discussion.

We asked the three organizations to invite the most relevant people to attend each group interview. This involved people with responsibility for the topic, for example from team level, subproject level and programme level. At *Accenture,* the key persons working on architecture were no longer working at the company. Table 2 gives an overview of the interview groups and lists the number of participants for each group. In total, 24 people participated. At *Steria* and the *Pension Fund,* some participated in several groups, making a total of 19 individuals. These people had a number of roles in Perform, from project management, subproject management, technical architecture, functional architecture, testing, development leadership, GUI, information security, Scrum masters, and developers (See Table 2). Note that there is a bias in participation towards employees in management positions (discussed in the limitations section). However, many of the participants had started in the programme as developers and assumed other roles during the programme. The participants selected where the ones who had

[5] Now Sopra Steria.

Table 2 Participants in group interviews and interviews. In total, 24 project members participated in interviews

Theme	Organisation	Participants	Roles present
Customer involvement	Accenture	1	Functional architect
	Steria	6	Functional architect, GUI responsible, subproject manager, technical architects, test responsible
	Pension Fund	3	Functional architects
Software architecture	Accenture	0	
	Steria	3	Technical architects
	Pension Fund	3	Chief architect, technical architects
Inter-team coordination	Accenture	2	Subproject managers
	Steria	3	Functional architect, subproject manager, technical architect
	Pension Fund	3	Scrum master, subproject manager, technical architect

responsibility for the topics we discuss, but we do not claim to represent all views of stakeholders on the topics. At the time of the data collection, all participants were seniors, with at least four years of experience in development, architecture or management positions, and at least four years of experience from their own organization. Interviewees were informed that the study is registered by and performed according to specifications by the Norwegian Data Protection Official for Research, which regulates collection and use of personal data material.

Two researchers participated in each group interview, and there was one interview guide per topic (see Appendix A). We developed a timeline for the project, which was shown to the participants so that they could remember key events in the project. The researchers had roles of leading and moderating the discussion and taking notes. All interviews would start with participants explaining their role in the project. One meeting at *Accenture* included only one person and was conducted as a semi-structured interview (Myers and Newman 2007). One group interview at *Steria* included six persons, and we used brainstorming techniques to ensure contribution from everyone invited. Participants brainstormed on key events, on what they thought was conducted well in the programme within the topic of discussion, and on what they thought could be improved. All nine meetings lasted two hours and were recorded and transcribed. We took pictures of drawings on whiteboards. This produced a total of 247 pages of transcribed material. The transcripts were sent to participants for information and for comments.

Second were three documents: the official report after project completion, the internal experience report (both from 2012), and the quality assurance report for the project, which was written by an external consulting company in 2010. We chose to include these documents as they provide overall descriptions of the programme such as planned organization, processes and roles. The internal experience report provides main lessons learned by the programme, the quality assurance report provides an assessment of main risks. In total, these reports contained 277 pages of text.

3.2 Data Analysis

We imported all interview transcripts and documents into a tool for qualitative analysis and did a descriptive and holistic coding (Saldaña 2009) of the topic inter-team coordination. Three researchers read the relevant documents and did individual coding, and then we agreed on coding in joint meetings. Units of text ranged from sentences to whole pages and were coded into topics such as “Scrum of Scrums” and “Metascrum”. The results of this pilot was presented and discussed with the rest of the research team, and we set up two research teams with two persons each to code the material on customer involvement and software architecture. Given the various topics and backgrounds of researchers, the level of detail in the coding varied between the research teams.

As an example, the statement, "*On an overall level, the teams worked quite similarly after exchanging experience. So the Scrum boards were used quite similarly across the teams. There were differences in colours, but the function was quite similar*", was coded as “board discussions” and grouped with 37 other concepts under “inter-team coordination”. The following passage was coded as “responsibility”, "*it was important that when teams got user stories and were to detail test conditions … that everyone contributed to detailing, then it was sent to project business for approval, and when you had the stamp from the business resource, you had agreed on scope*". This was coded with six other concepts under the category “participation”, which was one of seven concepts under “practice”, which again was one of four main categories under “customer involvement”. In total, there were 29 concepts related to this topic and 17 concepts related to software architecture.

After coding the material, we discussed what the most interesting findings were in the material. This was then made into a slide presentation, and feedback meetings were conducted with all three organizations *Accenture, Steria,* and the *Pension Fund* for member checking. These meetings were recorded and partially transcribed to add further data material, which led to small revisions of the first findings. Reactions to findings included surprise with the number of coordination arenas identified in the study, and comments on the thorough description of the architectural work in the programme.

4 Results

We first provide a description of the Perform programme before presenting results on our four areas of data collection: programme organization, customer involvement, software architecture, and inter-team coordination.

4.1 The Perform Programme

The Perform programme was conducted by the public department the *Norwegian Public Service Pension Fund (the* “*Pension Fund*”*)*, which needed a new office automation system. The main arguments for initiating the programme were public reform that required new functionality in office automation, and the existing office automation system was on a platform that was to be abandoned. When the programme started, the content of the public reform was not known. This was the main reason for choosing agile development practices for the project. The public department has about 380 employees and provides 950,000 customers with several types of services. The department integrates heavily with one other public department.

Perform was initiated to enable the department to provide accurate and timely services to customers and ensure a cost-effective implementation of the reform. Because of the reform, the programme had a strict deadline. It is one of the largest IT programmes in Norway, with a final budget of about EUR 140 million. The programme started in January 2008 and lasted until

March 2012. The project involved 175 people, of whom 100 were external consultants from five companies.[6] The programme used both time and material and target price contracts for subcontractors. About 800,000 person-hours were used to develop around 300 epics, with a total of about 2500 user stories. These epics were divided into 12 releases as in Fig. 1.

As an example, release R8 contained coupling of workflow in the office automation system to the archive solution, a self-service solution for new legislation, simulation of services towards external public departments, and first data warehouse reports on new data warehouse architecture. Most user stories were identified prior to the first release but were supplemented and reprioritised for every release.

The existing office automation system was client/server-based and written in C. The new system is a service-oriented system written in Java and provides a richer interface using Flex.[7] The database from the old system was kept, but the data model was changed. The regulations and legislations are implemented in the new system as rules using JRules. The system is integrated with a new document archive and with systems from another public department. Figure 2 gives an overall description of the existing solution and the new solution.

The programme was managed by a programme director who mainly focused on external relations, a programme manager focusing on the operations, and a controller and four project managers responsible for the *business, construction, architecture,* and *test* projects (see Fig. 3):

- *Business* - Responsible for analysis of needs through defining and prioritizing epics and user stories in a product backlog. This project was manned with product owners and a total of 30 employees[8] from the line organization in the department. Functional and technical architects from development teams also contributed to this project.
- *Architecture* - Responsible for defining the overall architecture in the programme and for detailing user stories in the solution description phase. Consisted of a lead architect and technical architects from the feature teams. Suppliers *Accenture* and *Steria* participated on a time & material basis.
- *Development* - Divided into three subprojects: one led by the *Pension Fund* (6 teams) with their own people and people from five consulting companies, and the two other subprojects led by *Accenture* (3 teams) and *Steria* (3 teams). The feature teams worked according to Scrum with three-week iterations, delivering on a common demonstration day. The feature teams had roles such as Scrum master, as listed in Table 3. In addition to the 12 feature teams, the project had an environment team responsible for development and test environments.
- *Test* - Responsible for testing procedures and approving deliverables from the development teams. Consisted of a lead tester and test resources from development teams.

There were also projects for communication and adoption to prepare users for the new systems, in total six projects.

[6] *Accenture* and *Steria* were the main external consulting companies, with responsibility for their own subprojects on development as well as providing personnel to other projects.

[7] Flex was developed by Adobe as a framework for "expressive web application" and is now an open source product.

[8] The number of people involved in the project varied; we use numbers in the peak period in the project from 2009 to 2011.

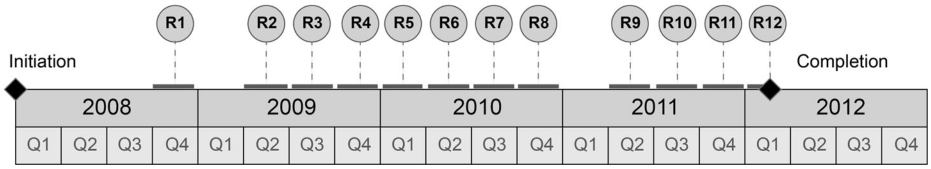

Fig. 1 Timeline for the Perform programme, showing 12 releases during the four-year period from initiation in January 2008 to completion in March 2012

As shown in Fig. 3, the programme used a matrix structure where the business and development projects took part in the architecture and test projects. This matrix structure meant that a feature team would then mainly take part in project *development*, while also devoting resources to projects *architecture* (through the technical architect), *business* (through the functional architect), and *test* (through the test responsible) as shown in Table 3.

The programme started working at the main office of the public department, but in 2009, it was moved to a separate office building located in the same city. Here, all teams were organized around tables as shown in Fig. 4.

Initially, the development process included the four phases described in Fig. 5:

- *Analysis of needs* - This phase started with a walkthrough of the target functionality of a release and identification of high-level user stories. Product owners prioritized the product backlog.
- *Solution description* - The user stories were assigned to epics, and the user stories were described in more detail, including design and architectural choices. User stories were estimated and assigned to a feature team.
- *Construction* - Development and delivery of functionally tested solutions from the product backlog. Five to seven iterations per release.
- *Approval* - A formal functional and non-functional test to verify that the whole release worked according to expectations. This included internal and external interfaces as well as interplay between systems.

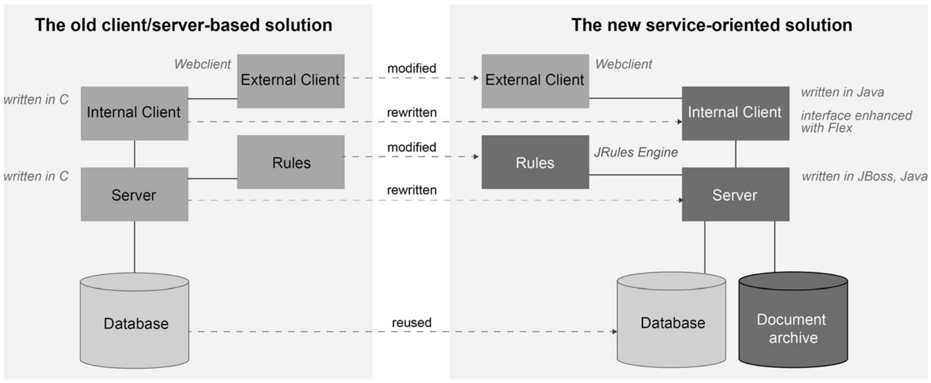

Fig. 2 Overall systems description of old and new solutions. The database was kept, but the data model was changed. The rules engine and the external web client were new

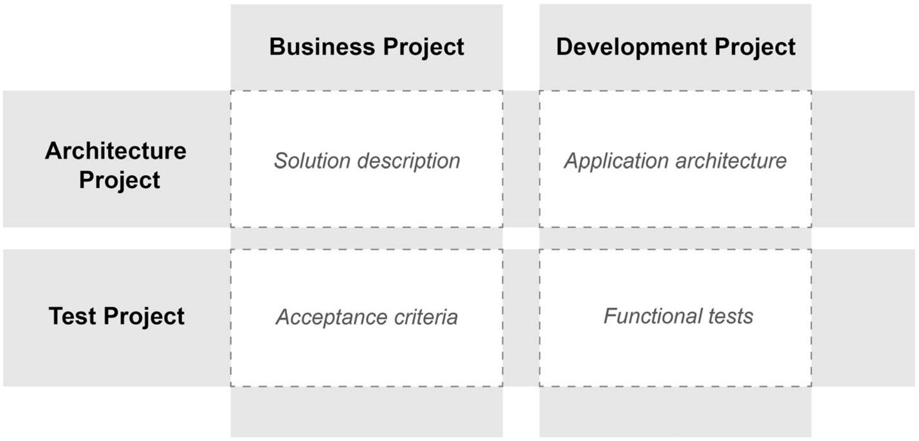

Fig. 3 The Perform Programme with four main projects: Business, Architecture, Development, and Test. The figure shows the matrix organization of the programme, with overlapping work between projects. 12 development teams worked in the programme at the peak. From 2009 to 2011 more than 150 people were working in the programme. The Development project was divided into three subprojects managed by the Pension Fund (6 teams), Accenture (3 teams) and Steria (3 teams)

To keep the schedule of the project, solution descriptions needed to be ready in time for the feature teams. This meant that releases were constantly being planned, constructed, and tested. Thus, given the roles in Table 3, a feature team would constantly be engaged in construction for release “n”, approving delivered functionality in release “n-1”, and analysing needs for the next release (“n + 1), as shown in Fig. 5. After approval from the programme, new releases were acceptance tested, set in production, and underwent an approval phase before being accepted by the operational IT section of the department.

4.2 Customer Involvement

The main arena for customer involvement in Perform was the interface between business and architecture subprojects and the analysis of needs and solution description phases. We found

Table 3 Roles in the feature teams and how the roles at team level connect to the main projects shown in Fig. 3

Role	Description
Scrum master	Facilitated daily meetings, iteration planning, demonstration and retrospective
Technical architect	Responsible for technical design, working 50% on this and 50% on development. Participated in project *architecture*
Functional architect	Responsible for detailing of needs. This role was usually allocated 50% to analysis and design, and 50% to development. Participated in project *business*
Test responsible	Made sure that testing was conducted at team level: unit tests, integration tests, system tests and system integration tests. Delivered test criteria to the project *test*
Developers	4–5 developers were allocated to a team (a mixture of junior and senior developers)

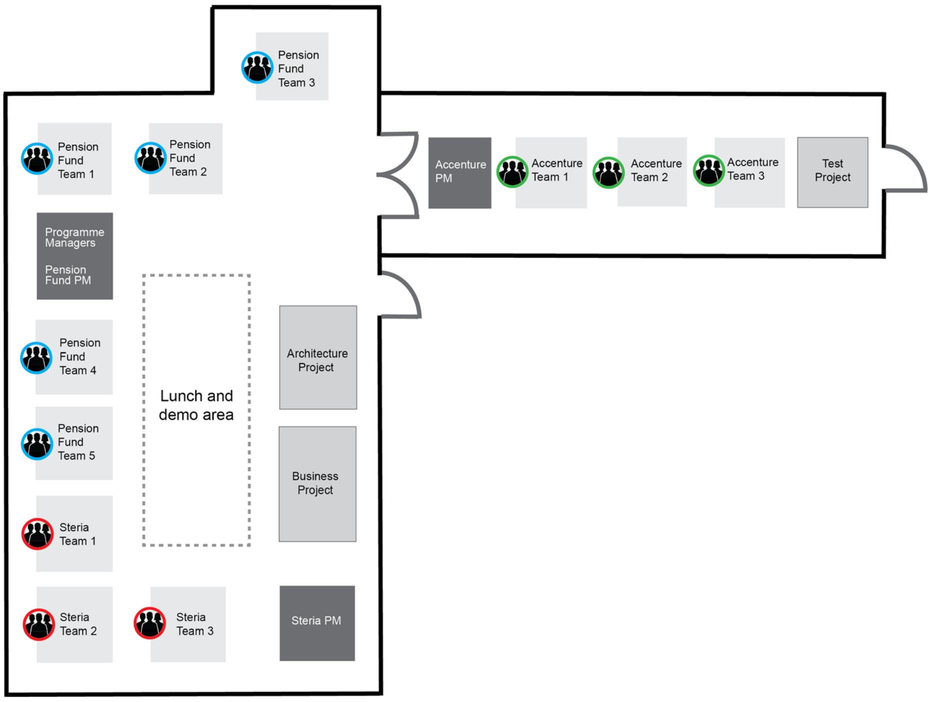

Fig. 4 Open work area where the development programme was situated from 2009 until project end in 2012. The figure shows the status when there were 11 development teams (Accenture 1–3, Steria 1–3, Pension Fund1–5). Accenture and Steria had separate project management tables, while Pension Fund's project management (PM) was located with programme management. Separate tables for projects business, architecture, and test

three key characteristics of this work: first, that the solution description was developed using teamwork; second, that it was conducted continuously and iteratively; and third, that the boundaries between analysis of needs, solution description, and construction were blurred. We describe each of these characteristics in the following.

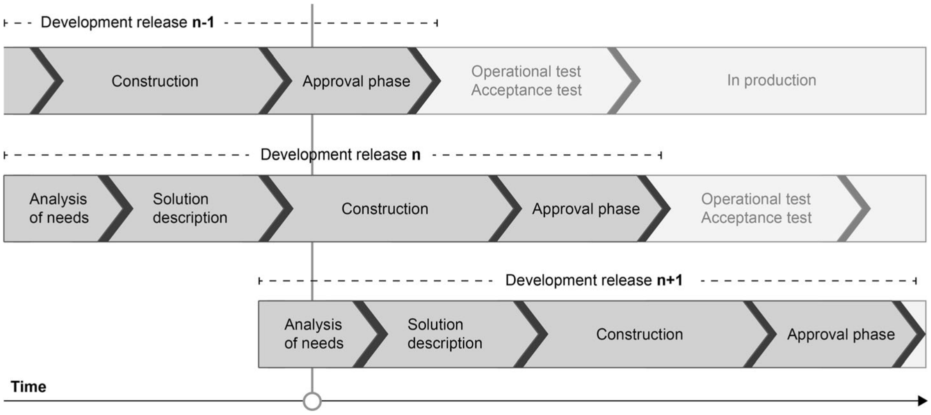

Fig. 5 Initial development process. From early in the programme, while working on release "n", the teams would also work with approving delivered functionality and analysing needs for the next release

4.2.1 Solution Description Teams

The solution description team included people who participated in the construction and people with relevant business skills from the customer organization. One informant described their role as being" *in front of the development team, before they start Sprint planning, we make a functional solution description with design of screens, error handling, services, conceptual and logical data model*" (functional architect). Membership could also span contractors: "*If it was only Steria that was involved, then it [solution description] was a business architect, the business representative, and some representatives from the Steria construction teams. If the work spanned contractors, we would involve people from Accenture, Steria, or the Pension Fund*" (technical architect). In addition, other people believed to be particularly knowledgeable for the work to be planned could be invited.

At least one member from the feature team (usually the functional architect) would participate in the solution description team. Solution description work was spread among construction team members: "*It wasn't one person's responsibility the whole time. It was natural for one to do solution description for a while and then to change it to another from the same team to ensure that everybody knew what we were working on*" (technical architect). This practice of involvement reduced the need for overly detailed solution descriptions. Another benefit was pointed out: "*It gave good continuity in the work and helped ensure that the construction was smooth and easy*" (functional architect).

This was seen as an effective practice for the program regarding knowledge sharing and collaboration, and was taken into account during iteration planning. However, it was challenging for the feature teams because they had to give up team members for a long period—often the most skilled and often in the last phase of the iteration.

4.2.2 A Continuous and Iterative Process

Solution description was a continuous process in two dimensions. It was continuous inside one release, and it was going on in several releases at the same time. Solution description for release n would start while construction for release n-1 was still underway (Fig. 5).

We learned that solution description work in Perform was characterized as "feeding the machine". The metaphor effectively conveys the property of the construction project as a machine that requires raw material (user stories) to produce products (construct the software system). The main purpose of solution description in Perform was to ensure that the product backlog contained enough high priority work to fill one to three iterations for the feature teams. Furthermore, it was essential to give the machine sufficient raw materials to avoid idle time. Most of the development project members were allocated full-time, and therefore, the cost of running the project "engine" would be constant, whether the backlog was empty or filled.

The detailing level in solution description was subject to change during the project. As the feature teams gained experience, they understood more about the problem domain and less detail was required to convey the essential parts of the tasks. We learned that solution description work became more efficient because "*they [the feature teams] understood what we wanted*" (functional architect). Informants expressed that it is much easier to communicate effectively when you know the audience. Furthermore, solution description was an iterative activity based on a pragmatic ambition to "*detail* *as much as time permits*" (functional architect). Hence, the level of detail was limited by the time available.

Tensions arose from working with a pipeline of completed solution descriptions with as few as one to three iterations. There were noticeable differences between the three suppliers *Accenture*, *Steria*, and the *Pension Fund* in terms of preferences for up-front planning. *Accenture* had a preference for more detailed, up-front specifications. *Accenture* wanted reduced risk: "*Regarding Pension Fund, we had to 'discipline' the customer somewhat, to clarify what they wanted before we took it into our list of committed work, so that we could get the least amount of rework*" (functional architect). *Steria*, on the other hand, was comfortable with more open specifications and continuous collaboration to resolve the details of the software in construction. The *Pension Fund* had more up-front descriptions than *Steria*, but less than *Accenture*.

4.2.3 Blurred Boundaries between Phases

Each release consisted of four consecutive phases: analysis of needs, solution description, construction, and approval. "*We could talk to anyone who might have the answer*" (technical architect). Customer representatives could approach the feature teams if they needed to discuss or clarify some issues. As a result, the boundaries were blurred between high-level requirements, solution description, and construction: "*In practice, high-level requirements and solution description merged more and more*" (another technical architect). This enabled more efficient collaboration. The same technical architect stated, "*As the contractor starts to understand more about the context, the customer's real problem is more visible, and this means we can find a solution together. Fast.*" Blurring the boundaries also gave another interesting result. It became possible for people working on solution description to question the output from the high-level requirement work. A contractor could challenge the *Pension Fund*: "*But could you do it this way instead? Why do it like this?*" (technical architect). Project participants found that this way of working stimulated creativity.

The iterative style of detailing the solution descriptions also contributed to blurring the boundaries. For example, the acceptance criteria were supposed to be defined in the solution description phase but were in practice often defined during construction by the teams. Our respondents were unable to agree on what phase the definition of acceptance criteria belonged to. Sometimes, the solution descriptions were too vague to enable construction and were passed back for revision.

We found pragmatism in the allocation of tasks to construction teams. We learned that often, the most challenging tasks were given to the *Pension Fund* feature teams because they had more domain knowledge and a more flexible contract model: "*If there were many changes, if there were complicated modules, and we didn't quite know how to design them, then the Pension Fund teams would get these tasks. I experienced the Pension Fund as being more flexible ... without the same constraints in relation to budgets and contracts*" (functional architect).

4.3 Software Architecture

Two topics emerged as primary characteristics related to managing work on software architecture: the tension between up-front and emergent architecture, and the demanding role for architects in large-scale agile projects.

4.3.1 Up-Front Architecture

In February 2009, a new chief architect entered Perform. At this point in time, the architecture of the new system was described in terms of architectural strategies but with little compliance in the system under construction. The core principle of the initial architecture was "service orientation", but it resulted in a complex system with many dependencies and high vulnerability in the event of changes. It was during this period that the decision was made to scale up Perform to more teams. The chief architect's most pressing concern was to create a system architecture that allowed the maximum parallelism of work in the different teams by "avoiding people stepping on each other's feet".

To account for this need, the chief architect and his team created a simpler system architecture focused around service modules and a horizontal unified "work surface" on top. The GUI layer—the "work surface"—made use of all the service modules. The architecture also established rules of ownership to specific parts of the database. An important benefit of the architecture was that it helped to separate the work of different teams into different parts of the code. In Perform, we found that the software architecture for the most part allowed the teams to work autonomously and effectively.

Even if this architecture manifested important decisions for the continued work in the teams, there were tensions stemming from having too little up-front work. Our respondents stated wishes for more "proof of concept" experiments before new technologies were adopted. A notable example was that the Flex technology used in the GUI layer had not been properly understood for it to be used effectively and establish sound "application architecture guidelines". Our respondents argued that the lack of proof of concepts was due to the contract model: "*This was partly governed by the payment model and the program's execution. If they had allocated resources for concept trials that possibly would have to be thrown away, or that resources had been allocated for the development of multiple candidate solutions—but resources were not allocated for that.*" (subproject manager, from feedback session).

At the end of the program, there was much work in optimizing the GUI layer code to obtain acceptable performance. Performance had been largely neglected for the main part of the programme but received significant attention in the final stages. This triggered some re-work for the architecture team: "*When we started to use this technology [Flex], we established an architecture that was fit only for small applications and small solutions. It was not appropriate for us when we had to fill it up with more logic and GUI elements*" (functional architect). Also, significant refactoring was done in the object-persistence layer[9] on top of the relational database management system for performance reasons. These refactoring efforts were packaged into specific user stories for non-functional requirements.

The programme put emphasis on common architectural principles: "*There needs to be openness with regard to suggesting architectural ideas from 'below' [in the teams], but you must also be able to convey merits of these ideas to convince the people in the appropriate forums so that it ends up as a unified architecture—so that each team does not use their own solutions*" (technical architect, feedback session).

The main architectural structure of the new system remained stable throughout the project. Mostly, the subsequent architectural work was local within the service modules.

[9] Using the Hibernate technology.

4.3.2 The Demanding Role of Architects in Very Large-Scale Programme

We learned that the architect role in very large agile projects is demanding because it requires continuous coordination and negotiation between many stakeholders, including representatives from the business side and the developers. The architect must be able to "sell" technically sound designs and technologies that may have short-term negative impacts on project progress and costs. Another tension was the approach to decision-making in the architecture project. For the most agile-oriented participants, this top-down decision-model came into conflict with their own ambitions to satisfy the business representatives. Team members pointed to a tendency of project architecture to step away from the discussions with business representatives when development teams pointed out that a particular architectural design had cost consequences for the business representatives. When developers argued to the business representatives that a proposed architectural approach would have penalties in terms of cost or performance, subproject architecture responded that it was unfair of developers "to hide behind the business people" instead of complying with architectural principles.

Architecture work was increasingly more difficult for project architecture as well. In the last half of the second year, the pressure to deliver functionality increased: "*At this point, project management and other management was concerned about progress ... It became much more difficult to promote improvements [in the architecture]*" (technical architect). "*Yes, more difficult. And then came these issues regarding technical debt, you know? Issues that are horrible in an agile setting*" (technical architect). Over time, technical debt would accumulate because the pressure to deliver functionality was so high. Due to the strict schedule in the programme, the business needs were often given priority to long-term architecture. Functionality was given priority over refactoring. We learned that team architects took steps 'on their own' to integrate refactoring into normal work tasks: "*For me it was equally important to establish the right attitude in the teams regarding code quality as it was to make the business side allocate resources for refactoring*" (technical architect, feedback session). We see that architectural work in agile projects is more continuous, and requires more integrative collaboration with a wider range of people.

The two suppliers *Accenture* and *Steria* used different approaches to architecture work during the project. *Accenture* made many architectural decisions during the solution description phase, while *Steria* delayed many such decisions until the construction. *Accenture's* motive was to ensure the highest possible efficiency of the teams' work during the Iteration. The culture in *Steria* appeared more flexible and collaborative, seeking rather to clarify issues as they emerged.

4.4 Arenas for Inter-Team Coordination

From our data material on coordination and knowledge-sharing between feature teams in the development programme, we identify three main findings. First, there were a number of arenas involved in order to achieve coordination and knowledge sharing. Second, the use of these arenas changed over time, and third, many emphasize the importance of the informal coordination arenas, which were enabled by the open work area. We describe results from each of these three areas in the following.

4.4.1 A Number of Coordination Arenas

Each feature team followed Scrum practices like having a planning meeting, daily stand-up meetings, retrospectives, and demos for each three-week iteration. There were a number of arenas in use to coordinate work and share knowledge between the teams, as shown in Table 4. This list includes mainly formal arenas such as the Scrum of Scrums and Metascrum at a level above the Scrum of Scrum. There were also several informal arenas such as experience forums, lunch seminars, and instant messaging. Some of the arenas were only used within a subproject, such as the "experience forum" at *Accenture* and the "technical corner" at *Steria*.

4.4.2 Changing Arenas Over Time

A characteristic of the Perform programme was that arenas for coordination changed over time. "*There were meetings that came and disappeared and some that remained*" (functional

Table 4 Coordination arenas used in the Perform project

Arena	Description
Board discussions	Every team had a board with tasks and status on one side and free space for drawing on the backside. The backside was used frequently for informal discussions
Demo	Demonstration of developed user stories after every iteration, from morning until lunch. Everyone could participate. 10 min devoted to each team. For rapid feedback, feature teams started organizing "mini demos" for participants in the business project within iterations
Experience forum	A meeting forum at subcontractor *Accenture* for Scrum masters, development manager and agile coach focusing on development method
Instant messaging	Instant messaging to all participants was set up after a need was identified in an open space session. Was used for open technical questions but also for social activities such as wine lottery
Lunch seminars	Seminar where 2–3 people gave short presentations during lunch on topics such as new architectural components, project management or on how to follow up on a team
Masterplan	The programme established a common product backlog as a master plan, with the 2500 user stories organized into 300 epics. The master plan was maintained in an issue tracker
Metascrum[a]	Two meetings per week with project managers from the development, architecture, test and the business projects, as well as subproject managers from the development projects
Open space technology	A process where all participants suggested topics for discussion, which is made into an agenda and participants are free to join discussion groups of interest. Used per release during parts of the project
Open work area	From 2009, the project with all teams and project management was situated in an open-plan office space on one floor
Retrospectives	Mainly used on team level, but a few times on subproject level. All retrospectives were documented in the programme wiki
Rotation of team members	Members sometimes rotated between teams, in particular in build-up phase when existing teams were split and new members added to all teams for a subproject
Scrum of Scrums	All three subprojects had Scrum of Scrum meetings with their teams, 2–3 times a week. Scrum masters and project manager attended, and sometimes others such as product owners and test managers
Technical corner	Architectural briefings in the subproject at *Steria,* where team architects briefed the teams. About 1.5 h, used in the beginning of the project
Wiki	Architectural guidelines, team routines, cross-team routines, system documentation, reports from retrospectives, solution descriptions and functional tests were documented on the project wiki

[a] Note that the forum Metascrum is used here with another meaning than the definition by the Agile Alliance, where "meta Scrum" and Scrum of Scrums are synonymous. Metascrums was at project level, while Scrum of Scrums were at subproject-level

architect). There were information needs that the programme saw was not covered, and new arenas such as the instant messaging were then established. Also, arenas were abandoned because the programme found that they were no longer useful. Open space is an example of an arena that existed for a while and then was abandoned. Some state that this arena had an effect: "*During a phase, I felt it had a mission, in particular to motivate people*" (subproject manager). Others were more critical of the practice: *" ...But in total, I feel that we did not get much out of it. Maybe we got to know each other better. Maybe.*" (another subproject manager).

Several commented on the change from formal mechanisms to informal mechanisms. Over time there was "*more usage of boards, more common coffee breaks, more lunches*" (subproject manager). Some arenas changed function over time. Internal meetings within the *Accenture* subproject developed over time into an arena for sharing technical knowledge. This illustrates that there was a change in both the arenas and in how the arenas functioned over time.

4.4.3 Open Work Area

Beginning in 2009, the project was co-located on an open-plan office floor. The teams were organized around tables, and project management for the whole programme and for the projects were on the same floor (see Fig. 5). Many stated that being in one open work area contributed to efficient coordination and knowledge sharing: "*It was the best possible solution that the project managers were at one table, in immediate reach of the development teams*" (subproject manager). Another subproject manager stated, "*I think being on the same floor is important. That is something I notice now being at [another large development programme], where we are not located on the same floor. It is much harder to know what is going on.*".

We learned that many of the decisions made in the project were discussed between relevant stakeholders informally and then officially decided upon in one of the formal arenas, the daily Scrums, Scrum of Scrums, or the Metascrum. Although mainly seen as vital to coordination and knowledge sharing, loud discussions on team tables in the open landscape led to some starting to use earphones in order to avoid noise and to indicate that they were not to be disturbed while working.

5 Discussion

Our research question is, "*How can agile methods be adapted in the very large scale, regarding programme organization, customer involvement, software architecture and inter-team coordination?*" We defined very large scale as projects or programmes involving more than ten development teams, with a high number of actors, and a large number of systems and interdependencies. The scale of such programmes leads to challenges including involving the customer, maintaining software architecture, and coordinating a large number of teams, projects, and subprojects. The Perform case shows some of the characteristics of the study by Batra et al. (2010), who describe making use of both traditional and agile methods. We give an overview of our findings in Table 5.

The *Pension Fund* decided to employ agile methods in the Perform programme because the content and extent of the public reform that the software system would support were unknown. We have provided a thorough description of the method adopted in the Perform programme and elaborated on some of the tensions that we propose are especially interesting themes for

Table 5 Main findings in the Perform programme

Area	Finding
The Perform programme	Matrix structure
	Development phases
Customer collaboration	Solution description teams
	A continuous and iterative process
	Blurred boundaries between phases
Software architecture	Up-front architecture
	Demanding role of architects in large programme
Inter-team coordination	A number of coordination arenas
	Changing arenas over time
	Open work area

future research on large-scale development projects. Based on our findings, we formulate research questions that should be addressed further in future research in three areas: customer involvement, learning and change, and inter-team coordination:

5.1 Customer Involvement

Agile methods describe the on-site customer to identify and prioritize features, provide feedback, and guide change through the course of the development (Vinekar et al. 2006). Customer involvement is a challenge in large-scale development efforts (Batra et al. 2010). Many representatives are needed, the representatives need to be aligned, and they must be available to all the teams. In large-scale projects the alignment challenge is considerable because different stakeholder groups have been found to have different priorities (Barney and Wohlin 2009). Further, a fragmented view of the system that each customer representative may have is likely to impact the project negatively (Ramesh et al. 2010), and the dependence on the on-site customer may cause project failure if the on-site customer is misaligned with the stakeholder goals (Vinekar et al. 2006). While each project in a large-scale effort may have success at the project level, there could still be overall failure at the program level if the projects are not aligned. The Perform program consisted of six subprojects and 175 people, which made alignment a challenge.

The issue of scale and availability was addressed by including 30 product owners and customer representatives from the *Pension Fund* that worked full time in the program. They were all in the "business" project and responsible for analysing needs through defining and prioritizing epics and user stories in a product backlog. Locating them in the same office space as the construction teams ensured their availability. Further, alignment among the on-site customers and between on-site customers and construction were supported throughout the whole program by 1) a number of arenas, 2) a continuous and iterative solution description process, and 3) the blurred boundaries between the phases of analysis and needs, solution description, and construction. Our findings are consistent with Barney and Wohlin (2009) who found that open and transparent dialogue and cross group communication as essential for aligning the different stakeholder groups.

Alignment within and across the six projects grew from the formal and informal arenas, continuous cooperation, and the constant opportunities to ask anyone about clarifications. Both customer representatives and developers could exchange and diffuse shared insights and knowledge through their network; that is, they could use their social capital (Wohlin et al.

2015) when aligning on all levels. Involving the teams in the solution description was important for the program success because it ensured alignment between teams and gave them end-to-end responsibility for features by involving them in the formulation of its requirement until the release. The importance of end-to-end responsibility in large-scale agile development is also consistent with Olsson et al. (2014) in their study of 30 teams in a large-scale program at Ericsson. Because there was always solution description work going on in the Perform program, the process can be understood as a continuous planning activity involving construction and business. Such continuous planning ensures alignment between the needs of the business and software development (Fitzgerald and Stol 2017).

While the programme under study enabled the concept of the on-site customer in practice, further research is needed to better understand the challenge of alignment between the on-site customer representatives and between on-site customer and construction in large-scale projects. Especially when co-location is not possible or feasible, if there is a high degree of uncertainty of what the customer wants, or there is a need for continuous deployment. We therefore propose the following research question for further investigation:

How can alignment be ensured among customer representatives and between customer representatives and developers in very large-scale programmes?

5.2 Learning on Architecture and Process

The system architecture and the development process are two vital areas of software development that could undergo changes during a development programme. For architects, it is difficult to balance concerns with delivering early customer value and taking care of overarching quality requirements (Hopkins and Harcombe 2014). The *Pension Fund* had to strike a balance and support both change and stability simultaneously (Vinekar et al. 2006). A system architecture defined up-front gave individual teams a stable working environment with room for teams to design their own local sub-architectures that probably contributed to team efficiency. Still, there were tensions from challenges that occurred because there had been few proof-of-concepts with core technology. With respect to the development process, many aspects remained unaltered. Team roles and iteration lengths are examples of this. Also, we found few changes in the larger-scale structures in Perform. The masterplan (Table 4)—the general matrix structure of the programme as well as the overall system architecture—remained largely unchanged during the programme period (Table 5).

Learning and change were evident at both the inter-team and programme levels in the Perform programme. At the inter-team level, we described that the coordination arenas changed over time, and there was a degree of autonomy regarding local architectural designs within the service modules. Further, deficiencies in information dissemination were met with active experimentation with new arenas for communication and coordination. Those arenas that did not bring value were abandoned. The open work area supported fast communication in informal meetings. Another example of learning that led to change is that the level of detail in solution descriptions was gradually reduced. The mutual understanding of the domain grew so that solution descriptions could be expressed more succinctly. At the programme level, Perform had mechanisms in place that enabled learning through numerous arenas for coordination and knowledge sharing. Perform gradually transformed towards an organizational culture with more trust and collaborative relationships. For example, the contractors took on a more collaborative role later in the programme where they were free and invited to question the assumptions underlying *Pension Fund's* requirements.

Perform was a learning-centric programme, yet its overall architecture and main development process remained unchanged for the bulk of the programme's duration. It is unclear how a more technically complex software system would impact the use of agile methods at such a large scale. An interesting research question is:

How can large-scale agile programmes enable learning and change in architecture and processes at inter-team and programme levels?

5.3 Inter-Team Coordination

Self-managing teams offer potential advantages over traditionally managed teams because they bring decision-making authority to the level of operational problems and increase the speed and accuracy of problem solving (Moe et al. 2009). In large projects, there are a number of problems that span development teams as well as decisions that affect a number of teams. Perform found a balance between self-management and a centralized structure. In the results section, we described how the matrix structure in the programme enabled coordination of development teams through roles in the teams and special arenas for the project's business, architecture, and test. This enabled direct communication between business and development, which we called blurred boundaries between phases.

A number of arenas in the early phase of the programme established communication and made it easy for team members to mutually adjust to activities in other teams. The open work area enabled quick oral coordination, but the extra roles (such as functional architect) and arenas (such as the Open Space) led to less autonomy for the team. The introduction of extra roles is a threat to self-management as the teams are instructed to establish certain roles, which Hackman (1986) refers to as managing their own work process. Further, teams needed to provide resources for projects other than development and had to engage in and accept decisions made in the projects that affected all team members. An example is the architectural guidelines and the overall architecture, which we have described as mainly being decided up-front. The standardization of roles and arenas across the project was primarily done to ensure needs at the programme level.

Contrary to findings reported by Paasivaara et al. (2012), we found that the Scrum of Scrums was perceived as a well-functioning coordination mechanism, but we could speculate that this could be because of the size or the focus on a large number of total arenas. We did not find use of Communities of Practice, which is described as a successful characteristic of an even larger development effort at Ericsson (Paasivaara and Lassenius 2014). We do not have insight on the total effectiveness of the coordination arenas in the Perform programme beyond knowing that the programme was recognized as a success. We could speculate that the dynamic nature of coordination with a large number of arenas made coordination efficient (Jarzabkowski et al. 2012).

It seems the programme found a good balance between inter-team coordination and self-management. However, this is a topic that deserves further research in describing other approaches and also examining the matrix structure applied in the Perform case in further detail. We pose the following research question for further investigation:

How do multi-team development programmes balance inter-team coordination and self-management?

5.4 Limitations

We use the most relevant principles for interpretative field research from Klein and Myers (1999) in discussing limitations of this study. Given the revelatory nature of this study, we have not put emphasis on limitations related to theory.

A main challenge in studying very large-scale programmes is the size of the case, and thus a challenge to understand both whole and parts (principle of hermeneutic circle). We have sought to address this by building an understanding of the whole through a thick description of method adaptation, and building understanding of three critical themes in an embedded case study design. Given the choice of three themes, there are a number of other areas that we have left unexplored, such as how quality was handled at the very large scale.

We have sought to satisfy the principle of contextualization by providing a thick description of the adapted development method in the programme, describe the development of key ideas on agile development in small teams in the theory and show how these ideas are present in the very large programme in presenting the empirical results.

Another limitation is related to the interaction between researchers and subject. We relied on group interviews and documents and as two data sources. For the group interviews, there is a potential elite bias (Myers and Newman 2007) in our data collection as the people involved in the programme at the end had technical or leader roles. We have avoided asking questions to evaluate the programme outcome but asked participants to reflect critically on practices and concrete events. We also had fewer participants from *Accenture* than from the other companies, which could have given us less variety of opinions. Further, the internal authors of documents could have an interest in portraying the programme as successful.

With the focus on three themes, we have aimed at identifying multiple interpretations by the different groups who participated in the programme, for example how architects and business project participants had different opinions on what was important at stages in the programme.

To discover biases and distortions in our findings (principle of suspicion), we have separated group interviews and presentation of findings to *Accenture,* the *Pension Fund* and *Steria.* Further, we have conducted workshops between researchers where all have had particular insight in one of the three main topics, thus sought to identify misconceptions from several angles.

Table 6 Items from research agenda (Dingsøyr and Moe 2014) and suggested research directions based on findings in the Perform programme

Research agenda	Suggested research direction
Customer collaboration	How can alignment be ensured among customer representatives and between customer representatives and developers in very large-scale programmes?
Knowledge sharing and improvement	How can large-scale agile programmes enable learning and change in architecture and processes at inter-team and programme levels?
Inter-team coordination	How do multi-team development programmes balance inter-team coordination and self-management?

6 Conclusion

The first main contribution of this study is a thorough description of a very large development programme that used a combination of agile and traditional methods. We believe this is an important contribution for practitioners seeking to conduct very large programmes in order to qualify advice in approaches such as agile methods. Our description of the Perform programme shows one model to organize very large-scale development, which to our knowledge is by far the most rich and detailed description available. We show how agile methods are adopted with additional processes such as the ones focusing on customer involvement and software architecture, and how inter-team coordination was organized with a number of extra arenas and extra roles at feature team level. This description will be valuable for students to learn how methods have been adopted to a specific context in very large scale.

The second contribution is showing some phenomena in organizing large programmes that are challenging at a very large scale. For the research community, we suggest questions to explore in the future in Table 6, and we hope that there will be more case studies to provide industry practitioners with research-based advice that can complement the practitioner-based advice that is available today. Topics of interest include how practices are adopted to scale, and how the context of development programmes influence the scaling approach. In future studies of very large-scale development programmes, we hope that researchers will be able to perform longitudinal studies to focus on variations over time and use a larger set of data collection methods such as participant observation to address the limitations that we have listed.

Acknowledgements We are very grateful to all participants in the group interviews. Further, we would like to thank colleagues Svein Hallsteinsen for taking part in data collection and analysis of software architecture and Tore Dybå for taking part in data collection and analysis of inter-team coordination, and both for taking part in discussions of the case. Thanks to master student Espen Andreassen at the Norwegian University of Science and Technology for conducting an independent analysis on parts of the material used in this article. We also thank Knut Rolland at SINTEF and Saskia Bick at the University of Mannheim for comments on earlier versions of this manuscript. Darja Smite at SINTEF provided great assistance in redesigning the figures to improve clarity. We would also like to thank anonymous reviewers for a number of suggestions for improvements. The work was supported by strategic internal projects at SINTEF on large-scale agile development and the project Agile 2.0 supported by the Research council of Norway through grant 236759 and by the companies Kantega, Kongsberg Defence & Aerospace, Sopra Steria, and Sticos.

Appendix: Interview Guides

Customer Involvement

<Introduction>

Which role and tasks did you carry out in the programme?
What was your responsibility?
<Timeline exercise to place main events>
Brainstorm: What worked well in the programme? What could have been better?
How do you define "solution description"?
What were the inputs and outputs to the "solution description"?
Who was involved in solution description and what were their roles?
What activities were carried out?
When were these activities carried out?

How did you coordinate work?
What kind of decisions were made?
Which role did the customer have in these decisions?
How were participants from construction involved?
How were architects and others involved?
How did you handle uncertainties in requirements and technology?

Software Architecture

<Introduction>
Which tasks did you have in the programme?
How would you describe the overall architecture in the programme?
<Timeline exercise focusing on the architecture>
How do you define "software architecture"?
How do you see the relationship between architecture and product characteristics?
What triggered events in the timeline with respect to architecture? (practices? meetings? decisions?)
Who were the architects in the programme (static role/dynamic?)
Who were the main stakeholders?
How were decisions made? Who made decisions?
How were trade-offs handled?
Who gave premises for architectural work? How did you obtain "input" for architectural work?
How was the architecture documented?
Did architectural decisions have an impact on the programme?
How was architecture communicated?
How did developers relate to the architecture?
What main architectural decisions were made?
Were there changes to the architecture over time?

Inter-Team Coordination

<Introduction>
Organization
* Draw teams and communication arenas between teams
* Who was sitting where?
Timeline exercise
* Important events in the project (in general).
Retrospective
* What worked well?
* What was challenging?
Inter-team coordination:
* How was the work organized in your part of the project?
* What kinds of dependencies were there between the teams in your part of the project? (examples?)
* How were dependencies managed? (examples?)

* What was managed in established fora and what was managed outside of the fora? (examples?)
* Who were involved in managing dependencies between teams? (examples?)
* Did you encounter challenges with managing dependencies? (examples?)
* Did you change the way you managed dependencies during the project? (examples?)
* What practices do you think were most important in order to manage dependencies between teams? (examples?)
* Are there any practices you think had little importance for managing dependencies?
* How did the division of the project into three main parts influence the coordination between teams?
* Were there differences in inter-team coordination across the subprojects?
* Scrum of Scrum
* Metascrum
* Architecture forum
* Test forum
* Requirement forum
* Open space meetings
* Unofficial fora? Direct contact
* Documents?
Frequency of meetings/persons involved/time spent
Knowledge sharing:
* What type of knowledge was important to share between teams in this project?
* What kinds of practices were used to share knowledge?
* How would you evaluate the different practices used?
* Was there any knowledge that was difficult to share?
* Is it possible to identify "knowledge communities" within the project?
* Which of these communities were official and which were unofficial?
* How was the work organized within the communities?
* How was knowledge sharing influenced by having several companies working together on the project?
* Which knowledge sharing arenas would you argue were most important for the project?
* Did a team know what other teams were working on?
* Were you able to shift workload across teams? Why was this easy/hard?
* Tool support?

References

Abrahamsson P, Babar MA, Kruchten P (2010) Agility and architecture: can they coexist? Introduction. IEEE Softw 27:16–22

Barlow JB, Giboney J, Keith MJ, Wilson D, Schuetzler R, Lowry PB, Vance A (2011) Overview and guidance on agile development in large organizations. Commun Assoc Inf Syst 29:25–44

Barney S, Wohlin C (2009) Software product quality: ensuring a common goal. In Wang Q, Garousi V, Madachy R, Pfahl D (eds) Trustworthy software development processes. ICSP 2009. Lecture notes in computer science, vol 5543. Springer, Berlin, Heidelberg

Bass JM (2015) How product owner teams scale agile methods to large distributed enterprises. Empir Softw Eng 20:1525–1557

Batra D, Xia W, Vander Meetr D, Dutta K (2010) Balancing agile and structured development approaches to successfully manage large distributed software projects: a case study from the Cuise line industry. Commun Assoc Inf Syst 27:379–394

Begel A, Nagappan N, Poile C. Layman, L (2009) Coordination in large-scale software teams, in *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, pp. 1–7

Bjarnason, E., Wnuk, K., and Regnell, B., (2012) "Are you biting off more than you can chew? A case study on causes and effects of overscoping in large-scale software engineering," *Inf Softw Technol,* vol. 54, pp. 1107–1124, 10// 2012.

Blichfeldt BS, Eskerod P (2008) Project portfolio management – There's more to it than what management enacts. *Int J Proj Manag* 26:357–365 5// 2008

Boehm B (2002) Get ready for agile methods, with care. IEEE Computer 35:64–69

Boehm B, Turner R (2003) *Balancing Agility and Discipline: A Guide for the Perplexed*: Addison-Wesley

Conboy K (2009) Agility from first principles: reconstructing the concept of agility in information systems development. Inf Syst Res 20:329–354 Sep 2009

Daneva M, van der Veen E, Amrit C, Ghaisas S, Sikkel K, Kumar R, Ajmeri N, Ramteerthkar U, Wieringa R (2013) Agile requirements prioritization in large-scale outsourced system projects: an empirical study. J Syst Softw 86:1333–1353

Dingsøyr, T. and Moe, N. B., (2014) "Towards Principles of Large-Scale Agile Development: A Summary of the workshop at XP2014 and a revised research agenda," in *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation*. vol. 199, T. Dingsøyr, N. B. Moe, S. Counsell, R. Tonelli, C. Gencel, and K. Petersen, Eds., ed Berlin: Springer, pp. 1–8.

Dingsøyr T, Nerur S, Balijepally V, Moe NB (2012) A decade of agile methodologies: towards explaining agile software development. J Syst Softw 85:1213–1221

Dingsøyr T, Fægri T, Itkonen, J (2014) What Is Large in Large-Scale? A Taxonomy of Scale for Agile Software Development, in *Product-Focused Software Process Improvement*. vol. 8892, A. Jedlitschka, P. Kuvaja, M. Kuhrmann, T. Männistö, J. Münch, and M. Raatikaineen, Eds., ed: Springer International Publishing, pp. 273–276

Dybå T, Dingsøyr T (2008) Empirical studies of agile software development: a systematic review. Inf Softw Technol 50:833–859

Eckstein J (2004) Agile software development in the large. Dorset House, New York

Eckstein, J., (2014) "Architecture in Large Scale Agile Development," in *Agile Methods: Large-Scale Development, Refactoring, Testing, and Estimation*. vol. 199, T. Dingsøyr, N. B. Moe, R. Tonelli, S. Counsell, C. Gencel, and K. Petersen, Eds., ed Berlin: Springer-Verlag Berlin, 2014, pp. 21–29.

Faber R (2010) Architects as service providers. IEEE Softw 27:33–40 Mar-Apr 2010

Fitzgerald B, Stol K-J (2017) Continuous software engineering: a roadmap and agenda. J Syst Softw:176–189

Fitzgerald B, Hartnett G, Conboy K (2006) Customizing agile methods to software practices at Intel Shannon. Eur J Inf Syst 15:200–213

Flyvbjerg B (2014) What you should know about megaprojects and why: an overview. Proj Manag J 45:6–19

Freudenberg S, Sharp H (2010) The top 10 burning research questions from practitioners, *IEEE Software,* pp. 8-9, 2010.

Giezen, M., (2012) "Keeping it simple? A case study into the advantages and disadvantages of reducing complexity in mega project planning," *International Journal of Project Management,* vol. 30, pp. 781–790, 10// 2012

Gregory, P., Barroca, L., Sharp, H., Deshpande, A., and Taylor, K., (2016) "The challenges that challenge: Engaging with agile practitioners' concerns," *Information and Software Technology,* vol. 75, pp. 26–38, 7// 2016.

Hackman JR (1986) *The psychology of self-management in organizations*: American Psychological Association

Hopkins R, Harcombe S (2014) Agile architecting: enabling the delivery of complex agile systems development projects, in *Agile Software Architecture*, M. A. Babar, A. W. Brown, and I. Mistrik, Eds., ed Boston: Morgan Kaufmann, pp. 291-314

Ingvaldsen JA, Rolfsen M (2012) Autonomous work groups and the challenge of inter-group coordination. Human Relations 65:861–881 Jul 2012

Jarzabkowski PA, Le JK, Feldman MS (2012) Toward a theory of coordinating: creating coordinating mechanisms in practice. Organ Sci 23:907–927 Jul-Aug 2012

Klein HK, Myers MD (1999) A set of principles for conducting and evaluating interpretative field studies in information systems. MIS Q 23:67–88

Kniberg H (2011) *Lean from the trenches: Managine Large-Scale Projects with Kanban*: the pragmatic bookshelf

Lagerberg L, Skude T, Emanuelsson P, Sandahl K, Ståhl D (2013) "The impact of agile principles and practices on large-scale software development projects: a multiple-case study of two projects at ericsson," in *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 348-356

Larman C, Vodde B (2013) "Scaling agile development," *CrossTalk,* p 9

Larman, C. and Vodde, B., (2017) *Large-Scale Scrum: More with LeSS*: Addison-Wesley professional.

Leffingwell, D., Yakyama, A., Knaster, R., Jemilo, D., and Oren, I., (2017) *SAFe reference guide: Scaled Agile Framework for Lean Software and Systems Engineering*: Addison Wesley.

Maiden N, Jones S (2010) Agile requirements can we have our cake and eat it too? Software, IEEE 27:87–88

Malone TW, Crowston K (1994) The interdisciplinary study of coordination. ACM Computing Surveys (CSUR) 26:87–119

Moe NB, Dingsøyr T, Dybå T (2009) Overcoming barriers to self-Management in Software Teams. IEEE Softw 26:20–26

Moe NB, Šmite D, Šāblis, A, Börjesson A.-L, Andréasson P (2014) "Networking in a large-scale distributed agile project," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, p. 12.

Myers, M. D. and Newman, M., (2007) "The qualitative interview in IS research: examining the craft," Inf Organ*,* vol. 17, pp. 2–26.

Nerur S, Balijepally V (2007) Theoretical reflections on agile development methodologies. Commun ACM 50: 79–83

Nerur S, Mahapatra R, Mangalaraj G (2005) Challenges of migrating to agile methodologies. Commun ACM 48: 72–78

Nord, R. L., Ozkaya, I., and Kruchten, P., (2014) "Agile in Distress: Architecture to the Rescue," in *Agile Methods: Large-Scale Development, Refactoring, Testing, and Estimation*. vol. 199, T. Dingsøyr, N. B. Moe, R. Tonelli, S. Counsell, C. Gencel, and K. Petersen, Eds., ed, 2014, pp. 43–57.

Olsson H, Sandberg A, Bosch J, Alahyari H (2014) Scale and responsiveness in large-scale software development. IEEE Softw 31:87–93

Paasivaara M, Lassenius C (2014) Communities of practice in a large distributed agile software development organization - case Ericsson. Inf Softw Technol 56:1556–1577 Dec 2014

Paasivaara, M., Lassenius, C., and Heikkila, V. T., (2012) "Inter-team Coordination in Large-Scale Globally Distributed Scrum: Do Scrum-of-Scrums Really Work?," in *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ed New York: IEEE, 2012, pp. 235–238.

Petersen K, Wohlin C (2010) The effect of moving from a plan-driven to an incremental software development approach with agile practices. Empir Softw Eng 15:654–693 Dec 2010

Rajlich V (2006) Changing the paradigm of software engineering. Commun ACM 49:67–70

Ramesh B, Cao L, Baskerville R (2010) Agile requirements engineering practices and challenges: an empirical study. Inf Syst J 20:449–480

Rolland, K. H., Fitzgerald, B., Dingsøyr, T., and Stol, K.-J (2016) "Problematizing agile in the large: alternative assumptions for large-scale agile development," in *International Conference on Information Systems*, Dublin

Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empir Softw Eng 14:131–164

Saldaña J (2009) The coding manual for qualitative researchers. Sage Publications Ltd, Second ed

Scheerer, A. and Kude, T., (2014) "Exploring coordination in large-scale agile software development: a Multiteam systems perspective," in *Thirty Fifth International Conference on Information Systems*, Auckland, 2014.

Scheerer, A., Hildenbrand, T., and Kude, T., (2014) "Coordination in Large-Scale Agile Software Development: A Multiteam Systems Perspective," in *2014 47th Hawaii International Conference on System Sciences*, R. H. Sprague, Ed., ed New York: Ieee, 2014, pp. 4780–4788.

Šmite D, Moe NB, Šāblis A, Wohlin C (2017) Software teams and their knowledge networks in large-scale software development. Inf Softw Technol 86:71–86

Smolander K (2002) Four metaphors of architecture in software organizations: finding out the meaning of architecture in practice. In: *Proceedings 2002 International Symposium on Empirical Software Engineering*, ed Nara. IEEE Computer Society, Japan, pp 211–221

Stettina CJ, Horz J (2015) Agile portfolio management: an empirical perspective on the practice in use. Int J Proj Manag 33:140–152 Jan 2015

Strode DE, Huff SL, Hope BG, Link S (2012) Coordination in co-located agile software development projects. J Syst Softw 85:1222–1238

Unphon, H. and Dittrich, Y., (2010) "Software architecture awareness in long-term software product evolution," *Journal of Systems and Software,* vol. 83, pp. 2211–2226, 11// 2010.

Vinekar, V., Slinkman, C. W., and Nerur, S., (2006) "Can Agile and traditional systems development approaches coexist? An ambidextrous view," *Information Systems Management,* vol. 23, pp. 31–42, Sum 2006.

Vlietland J, van Vliet H (2015) Towards a governance framework for chains of Scrum teams. Inf Softw Technol 57:52–65 Jan 2015

Wenger E (1998) Communities of practise : learning, meaning and identity. Cambridge University Press, Cambridge

Williams L, Cockburn A (2003) Agile software development: It's about feedback and change. IEEE Computer 36:39–43

Wohlin, C., Šmite, D., and Moe, N. B., 2015 "A general theory of software engineering: Balancing human, social and organizational capitals," *Journal of Systems and Software,* vol. 109, pp. 229–242, 11// 2015.

Yin RK (2014) *Case Study Research: design and methods*, 5th ed.: sage publications

Torgeir Dingsøyr focuses on software process improvement and knowledge management as chief scientist at the SINTEF research foundation. In particular, he has studied agile software development through a number of case studies, co-authored the systematic review of empirical studies, co-edited the book *Agile Software Development: Current Research and Future Directions*, and co-edited the special issue on Agile Methods in the Journal of Systems and Software. He wrote his doctoral thesis on *Knowledge Management in Medium-Sized Software Consulting Companies* at the Department of Computer and Information Science, Norwegian University of Science and Technology, where he is adjunct professor.

Nils Brede Moe works with software process improvement, intellectual capital, and agile and global software development as a senior scientist at SINTEF. His research interests are related to organizational, socio-technical, and global/distributed aspects. His publications include several longitudinal studies on self-management, decision making, innovation, and teamwork. He has co-edited the books *Agile Software Development: Current Research and Future*

Directions and *Agility Across Time and Space: Implementing Agile Methods in Global Software Projects.* His thesis was, “From Improving Processes to Improving Practice - Software Process Improvement in Transition from Plan-driven to Change-driven Development”. He holds an adjunct position at the Blekinge Institute of Technology in Sweden.

Tor Erlend Fægri works as a researcher at SINTEF, where he specializes in software process improvement and enabling practice-based group-wise learning. His research interests are in socio-technical aspects of learning and knowledge in software practice. His publications span learning practices in agile development and flexibility in software architectures. He wrote his thesis on collaborative learning in software development at the Norwegian University of Science and Technology.

Eva Amdahl Seim works with organizational and knowledge process improvement, agile software development, and ICT-based process and decision support as a senior scientist at the SINTEF research foundation. Her research is related to socio-technical, organizational, and project/portfolio aspects in ICT-related industries and traditional industries in general. Her publications include studies on project management, lean construction, and operational integration. She has participated in projects on creating real-time transparency and situation awareness in operational processes. She has a PhD in project management and has co-edited two books on interdisciplinarity in the knowledge of economics.