Agile Methodologies in Large Scale Information Systems Project Context – A Literature Review and Reflections

Katri Aintila

MSc Thesis

UNIVERSITY OF HELSINKI
Department of Computer Science

Helsinki, May 22, 2016

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta – Fakultet –Faculty Faculty of Science	Laitos – Institution – Department Department of Computer Science
Tekijä – Författare – Author Katri Aintila	
Työn nimi – Arbetets titel – Title Agile Methodologies in Large Scale Information Systems Project Context – A Literature Review and Reflections	
Oppiaine – Läroämne – Subject Computer Science	

Työn laji – Arbetets art – Level M. Sc. Thesis	Aika – Datum – Month and year 22/5/2016	Sivumäärä – Sidoantal – Number of pages 50 pages + 7 pages of appendices

Tiivistelmä – Referat – Abstract

Expected benefits from agile methodologies to project success have encouraged organizations to extend agile approaches to areas they were not originally intended to such as large scale information systems projects. Research regarding agile methods in large scale software development projects have existed for few years and it is considered as its own research area. This study investigates agile methods on the large scale software development and information systems projects and its goal is to produce more understanding of agile methods suitability and the conditions under which they would most likely contribute to project success. The goal is specified with three research questions; *I) what are the characteristics specific to large scale software engineering projects or large scale Information Systems project, II) what are the challenges caused by these characteristics* and *III) how agile methodologies mitigate these challenges?*

In this study resent research papers related to the subject are investigated and characteristics of large scale projects and challenges associated to them are recognized. Material of the topic was searched starting from the conference publications and distributions sites related to the subject. Collected information is supplemented with the analysis of project characteristics against SWEBOK knowledge areas. Resulting challenge categories are mapped against agile practises promoted by Agile Alliance to conclude the impact of practises to the challenges. Study is not a systematics literature review.

As a result 6 characteristics specific to large scale software development and IS projects and 10 challenge categories associated to these characteristics are recognized. The analysis reveals that agile practises enhance the team level performance and provide direct practises to manage challenges associated to high amount of changes and unpredictability of software process both characteristic to a large scale IS project but challenges still remain on the cross team and overall project level.
As a conclusion it is stated that when seeking the process model with agile approach which would respond to all the characteristics of large scale project thus adding the likelihood of project success adaptations of current practises and development of additional practises are needed.

To contribute this four areas for adaptations and additional practises are suggested when scaling agile methodologies over large scale project contexts; 1) adaptation of practises related to distribution, assignment and follow up of tasks, 2) alignment of practises related to software development process, ways of working and common principles over all teams, 3) developing additional practises to facilitate collaboration between teams, to ensure interactions with the cross functional project dimensions and to strengthen the dependency management and decision making between all project dimensions and 4) possibly developing and aligning practises to facilitate teams' external communication.

Results of the study are expected to be useful for software development and IS project practitioners when considering agile method adoptions or adaptations in a large scale project context.

ACM Computing Classification System (CCS) 2012:
• Social and professional topics~Management of computing and information systems • Software and its engineering~Software creation and management

Avainsanat – Nyckelord – Keywords
Large scale IS project, agile software development, large scale agile

Table of Contents

List of Abbreviations

AgileUP	Agile Unified Process
ATDD	Acceptance Test Driven Development
BDD	Behaviour Driven Development
CRC cards	Class, Responsibilities, Collaborators
CSF	Critical Success Factor
DSDM	Dynamic System Development Method
ER	Entity Relationships
FDD	Feature Driven Development
ICB	IPMA Competence Baseline
IPMA	International Project Management Association
IS	Information System
ISD	Information System Development
PMBOK	Guide to the Project Management Body of Knowledge
PMI	Project Management Institute
PRINCE2	PRojects IN Controlled Environments
RUP	Rational Unified Process
SAFe	Scaled Agile Framework
SWEBOK	Guide to the Software Engineering Body of Knowledge
TDD	Test Driven Development
UML	Unified Modelling Language
UX design	User eXperience design
XP	eXtreme Programming

1 Introduction

Fifteen years after agile manifesto agile software development methods are widely adopted by software development organizations and agile methods are utilized in various industries and various types of projects. Expected benefits from agile methods are attractive and there is evidence of the positive impact of agile methodologies to project success in terms of efficiency and overall stakeholder satisfaction [SeP15].

For these reasons organizations have extended the use of agile approaches to areas they were not originally designed to. Large scale software engineering projects or information system projects are one such area. Large scale IS projects are often highly business critical to the organization, their success is crucial since failing would have major impact on the business performance and it is especially tempting to utilise promising methodologies in such project contexts.

Agile methods in large scale projects have been recognized as a separate research area for few years now. It has been workshop topic in International Conference on Agile Software Development on years 2013 (XP2013) and 2014 (XP2014) and the Workshop on Large-Scale Agile Development is on the agenda for coming XP2016 [DiM13, DiM14, Laa14]. Also educational publications about the topic exist [LaV09].

The goal of this study is to investigate benefits and challenges of agile methods on the large scale software development and information systems projects. It aims on producing more understanding and concrete suggestions regarding the usage of agile methods in such contexts. Study includes analysis and resulting conclusions and propositions related to expected benefits, challenges and adaptation needs of agile methods in large scale software development or IS projects. It is targeted to software development and IS project practitioners and results are expected to be useful when considering agile method adoptions or adaptations in a large scale project context.

The study does not limit to software engineering discipline or software development lifecycle alone but is concerned of projects from the initiation to the project closure. It also does not limit to software engineering projects but is concerned of Information Systems projects. In this study Information System is considered being any organized system for processing information, which may or may not include technical components and software. Information System project is project involving creation or modifying information systems. While software engineering project can be considered limited to a software creation from the requirements definition to the complete installable software, Information System project may include creation of software or other technical components but moreover it may include other aspects such as modifying organizations processes and structures. Compared to software engineering project, Information Systems project includes dimensions such as business process modelling and design, management of change in terms of work instructions creation, trainings and communications, questions and answers and rollout to operational organization, which the possible software engineering dimension of the project needs to support. While most agile methodologies noted in this study are developed for the software engineering and the research literature inspected tends to limit to the software engineering aspect of the projects the methodologies are commonly used in the context of Information System project. For this both concepts are deliberately included in this study.

For more concrete definition of the study goal three research questions were set:

I. ***What are the characteristics specific to large scale software engineering projects or large scale Information Systems project?***

II. ***What are the challenges caused by these characteristics?***

III. ***How agile methodologies mitigate these challenges?***

Figure 1 presents the mapping of research questions to the study structure.

As back ground material reviews and studies of projects or programs including agile methodologies were searched from Scopus. Material was selected based on relevance estimated by reading the titles and/or abstracts, the focus being in articles addressing methodology selection or projects success. In the beginning of the search literature published after year 2000 was included, but the later selections limited to literature published after 2010 in order to both limit the search results and to concentrate on the most recent research. Systematic reviews and studies including large material bases (multiple cases or otherwise large samples of statistics) were preferred. Study is not a systematic literature review.

To answer the research questions I) and II) recent research literature about agile development methodologies in large scale contexts were inspected to find out what are different definitions of large, the characteristics typical to large scale context and the problems associated to these characteristics. The material search concentrated on the recent conferences where the subject has been raised as a topic; International Conference on Agile Software Development on years 2013 (XP2013) and 2014 (XP2014). Material was searched from the conference distribution sites and conference publications, in addition publications referred in the selected material were investigated as possible additional sources. Amount of research papers found about scaling was scarce, which was also noted in some of the papers included. To get more understanding complete analysis of the challenges was then done using SWEBOK knowledge areas [SWE14] as a framework against which to consider found characteristics of large scale IS project. To answer research question III) the challenges resulting from this analysis were then compared against agile practises to see how various agile practises used in different agile methodologies respond and mitigate these challenges.

Scaling agility over large scale organization other than project context, e.g. scaling over whole enterprise, is excluded from the study. This is also the reason why SAFe (Scaled Agile Framework) is excluded from this study although it was considered as possible methodology.

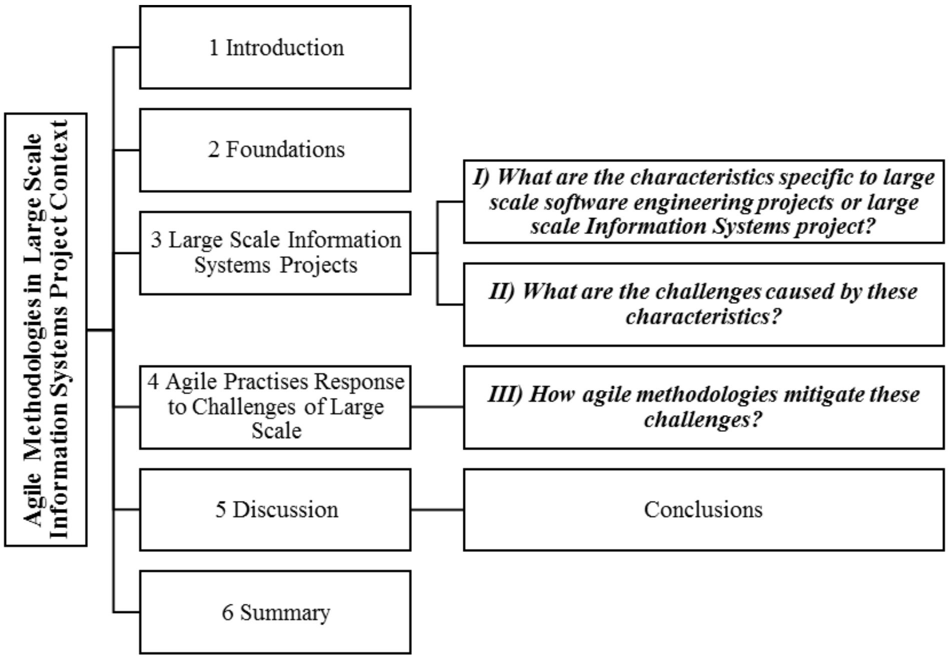
Agile Methodologies in Large Scale Information Systems Project Context
1 Introduction
2 Foundations
3 Large Scale Information Systems Projects
4 Agile Practises Response to Challenges of Large Scale
5 Discussion
6 Summary
I) What are the characteristics specific to large scale software engineering projects or large scale Information Systems project?
II) What are the challenges caused by these characteristics?
III) How agile methodologies mitigate these challenges?
Conclusions

Figure 1: Research questions are addressed in chapters 3 and 4 and conclusions are explained in chapter 5.

Chapter 2 introduces core concepts and background from research regarding different methodology approaches and project success and failure factors. Subchapter 2.1 explains plan based and agile methodology approaches and their main differences. Subchapter 2.2 refers how project methodology is considered impacting to project success.

Chapter 3 and its subchapters contain the recognition of the large scale project characteristics, the associated challenges and their categorization. Subchapter 3.1 explains different views on how large scale is understood in the literature, the characteristics and challenges of large scale projects collected from the literature are presented in subchapters 3.2 and 3.3. Subchapter 3.4 contains analysis of the impact of large scale project characteristics to software engineering project related knowledge areas. The analysis of agile practises impact to challenges of large scale IS projects is described in chapter 4 and its subchapters. Subchapter 4.1 presents the agile practises used in the analysis. Impact of agile practises to challenges in large scale projects is analysed in chapter 4.2 and the analysis results are explained in chapter 4.3. Results are summarised in Chapter 4.4 and further explained, concluded and discussed in chapter 4.5. Final conclusions are presented in chapter 5.

2 Foundations

Following chapters present the core concepts; *plan based* and *agile* methodology approaches, project *success* and *failure factors* and impact of methodology on the project success or failure as it is argued on the existing research literature.

2.1 Two Categories of Methodology Practise

Two major categories can be recognised from the current vendor communities of methodology practise related to software development projects; traditional plan-based and agile [ACD15]. Plan-based and agile approach differ greatly in terms of life cycle model, level of uncertainty and attitude towards changes. Plan-based approaches usually accommodate lifecycle models which are linear sequential or incremental and phased by scope, rely on pre-established plan and expect conformance to it considering changes as exceptions and disturbance that should be prevented. Instead agile approaches utilise iterative and adaptive lifecycle with short time boxes to allow learning from the feed-back and reprioritization. Since change is expected planning is kept short term and future features are not prepared in advance, modifying previous work and reprioritizing is allowed [ACD15, BTB03, DyD08]. Comparison of the two methodology categories is presented in Table 1.

Notable communities representing plan-based approaches are for example Project Management Institute (PMI), which publishes PMBOK®, International Project Management Association (IPMA), publishing IPMA Competence Baseline framework and PRINCE2® (Projects IN Controlled Environments), originally established by Office of Government Commerce UK, now days de facto standard developed and used extensively by the UK government, a registered trade mark of AXELOS Limited [PMB13, ICB06, MSP09]. PMBOK Guide fifth edition includes also Software Extension, developed jointly with IEEE Computer Society concentrating on management of software development projects, which is stated to bridge the gap between traditional and iterative e.g. agile approaches [SEP13]. Agile development principles and practises are promoted by Agile Alliance for the most [GtA15].

	Traditional Plan Based	**Agile**
Life cycle model	Linear sequential or incremental and phased by scope	Iterative and adaptive lifecycle with short time boxes
Level of uncertainty	Pre-established plan Conformance to plan expected	Learning from the feedback expected, modifying previous work and reprioritizing is allowed
Attitude towards change	Considering changes as exceptions and disturbance that should be prevented	Change is expected, planning kept short term, future features are not prepared in advance
Software development process methodologies	For example RUP	Scrum, XP, Scrum/XP Hybrid, Scrumban, Kanban, Iterative Development, Lean Development, Agile Modelling, Feature Driven Development (FDD), DSDM/Atern, XP, Agile Unified Process (AgileUP), Crystal, Custom Hybrid (multiple methodologies)
Guiding Principles	Plan based project management principles and guiding documents such as: Prince2® (Projects IN Controlled Environments) PMBOK® (Project Management Body of Knowledge) ICB (IPMA Competence Baseline)	Agile principles in Agile Manifesto

Table 1: Comparison of two categories of methodology practise.

It is to be noted that PMI PMBOK® and PRINCE2® are concentrating on project management process level, which are in software development context used together with the selected suitable software development process, (for example RUP, Rational Unified Process). In addition different software development related techniques (e.g. modelling languages; ER, process flow diagrams, UML) can be used on top of selected project management methodology and software process methodology. Instead, agile methodologies represented in the literature usually are software development methodologies which contain aspects of both project management level (for example dividing work into time-boxes impacts the schedule management on project management level, monitoring the work using burn down charts impacts on the scope, budget and schedule management) and development related techniques (documenting requirements as user stories). Agile methods focus on different aspects of the software development lifecycle such as management of the development, defining the development process or the practises and work products within the process, and they may cover different parts of the lifecycle [ASR02]. Most of the agile methodologies do not consider project management as a whole or cover other project areas such as project initiation, subcontracting, solution rollout and handover which are outside of the software development.

According to recent mapping study from Diebold and Dahlem, around 20 different agile or lean methodologies can be recognised but only small number of them are really used, most common being scrum and XP [DiD14]. State of the agile survey separates 11 agile methodologies; Scrum/XP Hybrid, Custom Hybrid (multiple methodologies), Scrumban, Kanban, Iterative Development, Lean Development, Agile Modelling, Feature Driven Development (FDD), DSDM/Atern, XP, Agile Unified Process (AgileUP) [SoA16]. In addition to these at least Crystal methodology family has been mentioned in the background material of this study [ChC08].

2.2 Project success and Project Methodology as one of the Success or Failure Factors

Definition of project success has evolved over time. Traditionally project success has been seen as conformance to a project plan, typically measured with attributes like budget, schedule and requirements [Yeo02] or similarly scope, time, cost and quality [ChC08]. In later studies this has been categorised as project management success [SAR12] or project process performance [ACD15]. Performance and quality of the product delivered as an outcome of the project have also been considered as attribute of the project success (categorized as project product performance) [ACD15]. Current studies related to software development projects state that there is no overall agreement over definition of success or universal success criteria that would be suitable for all projects [ACD15]. Project goals and expectations of different stakeholders impact on the perception of the project success and success criteria are therefore considered dependent on the project type and stakeholder perspective. For example customer satisfaction, short term business success (suppliers profit) and long term business success (future business, including good relations with customer) have been recognized as types of project success from software supplier perspective while meeting the planning goals (project management success), end user benefits (success from end user perspective) and contractor benefits (commercial success and potential for future revenue) have been recognized as success for research and development projects [SAR12].

Project success (Critical Success Factor, CSF) and failure factors are elements that are considered increasing the likelihood of success or failure [SAR12]. Project success/failure factors have been studied in both agile and traditional plan based methodology contexts.

Different sources or categories for success or failure factors have been proposed. For example Yeo has grouped failure factors as process driven (including business planning, project planning and project management/control related issues), context driven, (such as corporate culture, corporate management, users and politics related issues) and content driven (issues related to information technology, business process and system design, IT/IS professionals and knowledge sources in the project domain) [Yeo02].

Examples of success factors found in the agile project contexts are similar. For example Chow and Cao have proposed five different success factor categories [ChC08]; In their study of success factors contributing success attributes quality, scope, time and cost on agile project contexts they found evidence that technical and people factors (agile software techniques, delivery strategy and team capability and customer involvement) have heavy impact on project success and process and organizational factors (project management and project definitions process, management commitment, organizational environment and team environment) have some impact on project success but they found no evidence on project factors (project nature, project type and project schedule related factors) impacting to project success [ChC08].

As in these examples, project management methodology has been considered as one of the elements that have impact on project success in both agile and traditional project contexts. There are claims that choosing the appropriate project management approach is amongst the most critical success/failure factors. One of the recent studies on this area states that even though the categories are similar the actual success factors differ greatly and are even opposite for agile and plan based projects [ACD15]. For example factors like project planning, requirements and specifications changes, project team general expertise and monitoring and controlling have different role and meaning in plan based than in agile contexts which explains why they may be contributing the success in one approach but not in the other. Hence universal set of critical success factors across all methodologies is unlikely, the importance of each CSF varies for each methodology and the selected project process itself impacts on the success. Methodology should therefore be selected based on identified CSFs and the conditions on which the methodology would be likely to succeed. [ACD15]. Project characteristics impact on the suitability of development methodologies and management structures has been widely recognized and research of the area includes studies, tools and framework proposals for aiding on the selection of the appropriate process model [GuD15, Kel05].

3 Large Scale Information System Projects

Following subchapters explain different aspects of large scale software development or IS projects, the special characteristics of such projects and the categorized challenges associated to the characteristics.

3.1 Aspects of Large Scale

In the information systems project related literature large scale usually refers to the size of the application domain impacted (e.g. enterprise application projects where development scope includes several applications of the enterprise application domain) [VlV15, RaA14], size of project organization (or large development organization for product line) [TRA15, DyD15, RaA14, SHK14, PaP14, DFI14] or time scale of the project (projects taking several years) [DyD15]. In many cases these different aspects are related. It is common for example in the enterprise application domain that separate teams work with different applications causing larger organizational set up. When the application domain is wide, using large organization does not usually shorten the development time. Large development organizations are also often meant to stay long since the cost (effort and time) of setting up large organization and getting it properly working is usually high. While large scale software engineering projects have been recognized as separate research area, literature is still scarce and the criteria for considering project being large are not commonly well defined. Dingsøyr, Fægri and Itkonen [DFI14] have proposed a taxonomy with three levels from small scale (1 team) to large scale (2-9 teams) and very large scale (10+ teams) development projects based on the amount of teams and their impact to the coordination approaches required. They also state that costs, code size or number of requirements are not suitable criterion for determining whether project is large or not, since they are often dependent on the domain, tools and technology used, reusable code base and length of the project and therefore are not comparable measures between projects.

3.2 Special Characteristics of Large Scale Information System Projects

Even though the definition of large scale is not clear or unified, common characteristics can be recognized from the research of large scale information systems projects. Six typical characteristics presented in the following chapter were identified from the literature included in this study.

Large scale project set up is usually a multi-team system. Multi-team setup was referred to in all eight source articles about large scale development in agile context [GBT15, DFI14, DyD15, Pap14, RaA14, SHK14, Tra15, VlV15]. In software engineering research this usually means that project includes several development teams, e.g. scrum teams due to the amount of product areas and features. Organizational project context may also enforce multi-team set up, for example in enterprise application domain applications usually have dedicated development teams which may be outsourced to vendors and if the scope includes several applications, it naturally includes several teams. In addition large scale projects often include other areas than just software development, such as rollout, trainings, business transformation management etc. which are also represented in the organizational set up and need to interact with the development teams, this is common for example business transformations and architecture consolidation and replacement projects. This was the most commonly mentioned feature in the reviewed literature.

Distributed teams are very typical to large scale projects with multi-team settings. This was mentioned in five articles out of eight [GBT15, Pap14, RaA14, Tra15, VlV15]. Large organization may not easily fit into same premises. In addition enterprise application maintenance and development is often at least partially outsourced to application specific vendors. Usually in large enterprise application projects (business transformations, systems consolidations or replacements) at least part of the project is outsourced to an external software vendor which uses its own premises for development work.

It is common to large scale development that the scope contains features spanning over several systems and development teams. Two of the included articles specifically referred to large features split and distributed to different teams [Tra15, VlV15], in addition coordination of dependencies is brought up in one source study [SHK14]. In the enterprise application domain it is common that the business functionality is implemented by interacting features in several applications. Therefore business process changes, new business functionalities or replacement of applications usually require development in several interacting applications often managed by separate development teams. The occurrence of this kind of requirements is especially high in large scale development projects in the enterprise application domain, but there is similarity also for example to embedded systems development projects where there are dependencies to features developed to infrastructure by external parties.

Large scale projects are usually also alive long time. Large problem domain naturally takes long time to be covered and completed but there are also often lots of other areas in addition to the software development, such as pre-study and initiation activities and acceptance, deployment, transition and rollout activities, that may be needed prior or after the development activities, which may impact the project total timeline. Scaling over time is mentioned in two studies included in the source material of features of large scale projects [GBT15, DyD15].

Some characteristics specific to information systems projects in general can be expected to gain even more significance in large scale context and are therefore also worth mentioning.

In large scale environment information system architecture and software can have unlimited complexity, revisibility, flexibility and nonlinear behaviour. Capturing and modelling every possibly condition that may impact the behaviour of system (system including all interacting applications and other actors) is impossible in large scale context. Software development and especially problem solving process are unpredictable by nature. In the context of large scale environment the problem is rarely fully understood from the beginning and may be changing or more of it is gradually revealed while more details are uncovered and some parts of the problem solved. It is common that the problem is fully understood and the requirements fully defined only when the solution is defined and until that it may be impossible to say how close to completion the solution is. Complexity of IS architecture and software as a product and unpredictability of development process are both mentioned in two separate articles [DyD15, SHK14].

Features of large scale IS projects are presented in the table 2.

Features of Large scale IS projects	**Research articles where occur**
Multiple teams	[GBT15, DFI14, DyD15, Pap14, RaA14, SHK14, Tra15, VlV15]
Distributed teams	[GBT15, Pap14, RaA14, Tra15, VlV15]
Large features spanning over several systems and teams	[SHK14, Tra15, VlV15]
Long timespan	[GBT15, DyD15]
Complexity of IS architecture and software as a product	[DyD15, SHK14]
Unpredictable nature of development process	[DyD15, SHK14]

Table 2: Features of large scale information systems project.

3.3 Challenges Associated to the Features of Large Scale Projects Using Agile Methodologies in Literature

The challenges associated to agile methodologies used in multi-team setup are related to cross-team coordination [DFI14, DyD15, Pap14]. Additional forums (such as multiple scrum of scrums) are needed to ensure the coordination between teams which causes co-ordination overhead. When organization hierarchy deepens risk of knowledge silos increases [DFI14]. Added organizational structures contradict with the agile principles and careful balancing of additional and adapted methodologies is needed to keep benefits of agile methodologies still real. Concrete decisions and questions to be resolved are related practises such as what would be the optimal organizational design, whether to have multi-team or multiple backlogs, should all participate on multiple meetings or only single representatives, selecting suitable tools for large scale setting, and ensuring the organizational agility of the operational environment [Pap14]. While agile methodologies prefer and rely on organic and cognitive coordination types, in a large multi-team setups mechanistic coordination is needed. Cognitive coordination (share mental models and transactive memory systems) cannot be established in multi-team system without help of other types of coordination. Pure organic (mutual adjustment via interaction) coordination requires excessive amount of communication between all members of multi-team system and the communication overhead would make it impossible which is the reason for scrum of scrums settings. Choosing the coordination strategy and optimal mixture of different coordination types is needed in multi-team systems [SHK14].

Distribution of teams increases the challenges of multi-team system. Lacking face-to-face communication and physical access added with time zone differences means that even basic information sharing require using communication technologies. More sophisticated tools and working environment is needed to support distributed development and in the same time vulnerability of the infrastructure and development environment increases increasing the risk of environment related quality problems [RaA14]. Depending on the organizational setting and the distribution model the challenges concentrate on different areas of the project organization. In settings where the development teams are geographically distributed from product owners (outsourcing) the collaboration between product owners and development teams is challenging and needs additional supportive practises

[Tra15]. Choosing optimal collaboration model (e.g. collaboration via scrum of scrums or cross functional teams of which members are distributed geographically) for teams in distributed context needs to be resolved [VlV15].

Large features spanning over architecture need to be split and distributed to different development teams causing interdependencies between teams. Dependencies increase the need of coordination to align priorities, schedules, working practises and deliverables. Amount of dependencies have high impact on the predictability of delivery. All involved teams need to deliver on time and failure to do so impacts the work of all teams in next iteration (causing re testing of something implemented in the previous), having significant impact also on time to market and costs. Coordination and sharing the goals and policies between teams is not supported in the agile methodologies and does not happen naturally in multi-team environment and therefore additional mechanisms are needed. Typically agile teams such as scrum focus on internal backlog instead of the end to end features and may therefore have mismatching priorities. Alignment of working processes and policies (such as definition of done, start, finish and duration of the increments, test activities and test results are needed to accomplish end to end features. Visibility to the status of other teams work is required and preferably automated [VlV15]. In addition to inter-team coordination the visible progress of full end to end feature is often slow and visibility over progress and possible problems is easily lost. Large end to end features may block the development pipeline unnecessarily when several teams are engaged to it but waiting other teams to complete. Splitting the features properly to manageable size while keeping the dependencies in minimum needs to be resolved [Tra15].

Challenges associated to time aspect and project length are related to changes. Changes in the environment, market conditions, customer requirements and project goals are normal in the information systems projects. While project size and length increases, changes accumulate over time and over the large problem domain so high amount of change is expected in the large scale project. It is common that even requirements of already implemented features may change and for large features which take long time to be completed changes may come even during implementation. In the information system projects taking several years the future is uncertain and because of the changes relying on past experiences is not reliable [DyD15].

Complexity of IS architecture and software as a product in large scale contexts poses also challenges related to changes. While it is not possible to model all requirements/design in advance or to build a models which produce accurate results about the system's qualities, ability to adapt changes and roles and techniques oriented toward flexibility and learning are needed [DyD15]. The complexity of large scale software development problem domain produces incomplete and changing requirements and it also hosts complex interdependencies between the requirements and existing infrastructure and software stack [SHK14].

The unpredictable nature of problem solving and information systems development process makes advance planning difficult. It is impossible to reliably plan all task durations and schedules of complex problem solving cases or details of the solution in advance in all circumstances. Therefore flexibility and ability to adapt is highly needed [DyD15].

3.4 Analysis of Impact of Large Scale to Software Engineering Project Related Knowledge Areas

Since the evidence found in the literature review in previous sections about the impact of large scale to information systems projects is little, more complete analysis was done using SWEBOK knowledge areas [SWE14] as a framework against which to consider each characteristics of large scale IS project.

The knowledge areas of *Computing foundations*, *Mathematical foundations* and *Engineering foundations* were left out from the analysis by default. These knowledge areas have more to do with the project content and information systems solutions in the project scope than the process of developing which is the scope of this study.

Since this study is concentrating on project aspect of the software engineering *Software maintenance* knowledge area was considered only in the context of an ongoing large scale project, not as a continuous process outside of development project.

Moreover, *Software engineering process* knowledge area was not separately considered. More complete analysis on relation of large scale characteristics and software engineering process is expected as a result of this study and including it to the analysis as such would create a self-reference to the expected results.

Column “Other” was added to capture possible other considerations related to software engineering knowledge areas in a large scale project context not directly associated to characteristics found on the literature. Breakdown of analysis of SWEBOK knowledge areas against the characteristics of large scale projects is presented in the table 3.

SWEBOK Knowledge Areas	**Features of large scale software engineering or IS projects**						
	Multiple teams	**Distributed teams**	**Large & spanning features**	**Long time span**	**Software / IS complexity**	**Problem solving process nature**	**Other**
Software requirements	Agreeing on common definitions (cross team), Deciding the best organization structure for reqs definition process	Reqs negotiation, communicating requirements.	Splitting large features to smaller sub-features and making architectural decisions impacting widely in the system landscape Recognition of dependencies and boundaries regarding split features	Long time span increases the amount of changing reqs	Information system inherent complexity causes incomplete and changing reqs	Due to unpredictable nature of problem solving process requirements may stay incomplete and changing and requirements engineering activity can't be completed before late in the development phase.	Large amount of requirements and requirements sources/stakeholders that need to be involved and satisfied
Software design	Agreeing on common principles, Distribution of design tasks	Communicating design with distributed teams,	Design of interfaces/interactions related to split features. Communicating the design regards to split features and architecture decisions, Synchronizing the design work of split features	During long time span changes may be inflicted to designed or completed features	Incomplete and changing reqs cause design changes	Due to unpredictable nature of problem solving, requirements definition, design and implementation are intertwined and can't be completed before completion of development and approval of the feature	-
Software construction	Agreeing on the coding standards Dividing the implementation work to teams	-	Synchronizing the implementation work of split features Integration and integration testing of split features	During long time span changes may be inflicted to designed or completed features	Incomplete and changing reqs cause changes during implementation time	Due to unpredictable nature of problem solving, development completion time may be difficult to predict before it's com-	Validating ad confirming the results with many stakeholders

SWEBOK Knowledge Areas	Features of large scale software engineering or IS projects						
	Multiple teams	**Distributed teams**	**Large & spanning features**	**Long time span**	**Software / IS complexity**	**Problem solving process nature**	**Other**
						pleted with verification and approval. Even after approval defects can be found causing changes to the design and implementation	
Software testing	Distribution of testing responsibilities over teams and to common testing organization. Agreeing on the approval and completion criteria for deliverables	Communicating requirements, Communicating test results/incidents with distributed teams	Following up and coordinating completion of split features for testing, Organizing E2E testing of large split features involving experts of multiple teams	Keeping requirements up to date during long time span Defining verification and approval criteria for reqs changing during the time span	Keeping requirements up to date during long time span Defining verification and approval criteria for changing reqs Since not all conditions can be tested, it is difficult to decide readiness for approval	Keeping requirements up to date during long time span Defining verification and approval criteria for changing reqs. Time and needed test rounds for feature can't be predicted, scheduling the approvals are difficult.	Validating ad confirming the results with many stakeholders
Software maintenance	-	-	Agreeing the incident management and maintenance responsibilities over large features involving several subsystems and possibly several maintenance organizations	Long development project may be still ongoing while maintenance process needs to be set up and the interaction between these two needs to be planned (in re-	-	-	-

SWEBOK Knowledge Areas	Features of large scale software engineering or IS projects						
	Multiple teams	**Distributed teams**	**Large & spanning features**	**Long time span**	**Software / IS complexity**	**Problem solving process nature**	**Other**
				gards to functionalities changed in both work streams and timing of changes near releases) Creating the documentation for maintenance when lots of content from long development project and changes still coming.			
Software configuration management	Coordinating sw configuration status with multiple teams	Communicating software configuration status to distributed teams	Keeping the dependencies when planning releases and managing builds including large split features. Keeping software configuration working in situations involving split features Planning timing and meaningful content for releases.	Planning timing and meaningful content for releases while changes to implemented features may already be known	Due to late finalization of requirements release content may not be fixed until nearly release time	Due to late finalization of requirements release content may not be fixed until nearly release time	-
Software engineering management	Defining organizational setting which facilitates	Ensuring knowledge sharing	Coordinating schedules and deliverables over	Expected changes during long time span lower the	Due to IS domain complexity Final solution can't be	Due to unpredictable nature of software development	-

SWEBOK Knowledge Areas	Features of large scale software engineering or IS projects						
	Multiple teams	**Distributed teams**	**Large & spanning features**	**Long time span**	**Software / IS complexity**	**Problem solving process nature**	**Other**
	engineering processes and coordination over functional areas. Organizing decision making and right participants over multiple teams. Ensuring knowledge sharing. Monitoring the total progress.	over distributed teams Coordination of the distributed teams regards common milestones and target schedules. Communicating the progress of distributed teams.	split features. Monitoring progress and completion of split features and completion of the feature.	credibility of the plans created in the initiation phase Changes during long time span cause lots of re-planning. Measuring success of project after lots of changes is difficult	fully defined in the initiation phase hence not all coming activities are known in initial planning phase, causing incomplete plans (schedule estimates, resource needs, recognised work packages and tasks, etc). Incomplete plans require updating and re-planning.	work (problem solving) all activities needed in the design and implementation phases can't be recognized in the initial planning causing incomplete plans. Incomplete plans require updating and re-planning.	
Software engineering process	Selection and tailoring of processes and lifecycle models to support features of large scale project						
Software engineering models and methods	Agreeing on common modelling languages and methods to needed extent between teams	Tool support for sharing models and other deliverables with distributed teams	Shared models over split features and their boundaries required. Need to recognize what must what is critical to understand and be modelled	Updating and communicating updated shared models after changes		Updating and communicating updated shared models after changes	-
Software quality	Agreeing and sharing the same criteria and standards	Agreeing and sharing the same criteria and standards	-	-	-	Deciding when and how to measure quality when	Validating ad confirming the results with many stakeholders

SWEBOK Knowledge Areas	Features of large scale software engineering or IS projects						
	Multiple teams	**Distributed teams**	**Large & spanning features**	**Long time span**	**Software / IS complexity**	**Problem solving process nature**	**Other**
	for quality over teams	for quality over teams				end results and requirements are not known/fixed until late stage	
Software engineering professional practice	-	-	-	Personnel changes likely during long time span, learning time and group dynamics aspects may have impact when personnel is changing.	-	-	-
Software engineering economics	Making prioritization and scoping decisions and tools/component selections which have different impacts over multiple teams	Need to make decisions over offshoring/outsourcing	Prioritization of split features in the context of each part	Changing business goals and priorities are possible during long time spans which impact the project feasibility, scope and success	IS complexity and inability to model everything adds uncertainty in decision making	Unpredictable nature of software development adds uncertainty in decision making	Decision making is difficult with various stakeholders having contradicting objectives
Computing foundations							
Mathematical foundations							
Engineering foundations							

Table 3: Analysis of SWEBOK knowledge areas against the characteristics of large scale IS projects.

In the analysis, all topics under the each SWEBOK knowledge area were reviewed and the impact of each feature of large scale projects is considered. For example first topic of SWEBOK Software Requirements knowledge area is “Requirements Fundamentals”, having sub-topics: “Definition of a Software Requirement”, “Product and Process Requirements”, “Functional and Nonfunctionl Requirements”, “Emergent Properties”, “Quantifiable Requirements”, “System Requirements and Software Requirements”. The impact of large scale project feature multiple teams to this topic is that all teams must understand the requirements fundamentals similar way in order to be able to contribute to or use the same requirements base, hence there is need to agree a common definitions between teams. Second topic in Software Requirements knowledge area is the “Software Requirements Process”. Impact of multiple teams to this topic depends on how the teams are organized and whether the requirements process includes interactions between teams or is something within the team. So the challenge of deciding the best organization structure for requirements definition is recognized. The challenges or needs recognized in this way are then grouped under common problem categories.

Detailed grouping of atomic challenges to groups is presented in the table 8 (Appendix 1). Mapping of problem categories to SWEBOK Knowledge Areas and large scale project features is presented in table 9 (Appendix 9).

Found problem categories are summarized in the table 4 and the mapping of the categories to features of large scale IS projects is presented in table 5.

Problem category
1. Sharing the same understanding across large organization
2. Setting roles and responsibilities over multiple teams
3. Distributing and assigning tasks for multiple teams
4. Decision making over multiple teams
5. Communication over multiple / distributed teams
6. Coordination and dependency management over multiple / distributed teams
7. Dealing with changes and unpredictability
8. Dealing with large amount of "customers"/stakeholders
9. Interacting with parallel software maintenance (or other organizational processes)
10. Personnel/human resources and sourcing decisions e.g. offshoring/outsourcing and personnel changes

Table 4: Categories of problems associated to large scale information systems projects.

Features of large scale IS projects	**Challenges associated**
Multiple teams	**(1)** Sharing the same understanding cross multiple teams **(2)** Setting roles and responsibilities over multiple teams. **(4)** Decision making over multiple teams. **(5)** Communication over multiple / distributed teams. **(6)** Coordination and dependency management over multiple / distributed teams
Distributed teams	**(1)** Sharing the same understanding cross multiple teams: **(5)** Communication over multiple / distributed teams **(6)** Coordination and dependency management over multiple / distributed teams **(10)** Personnel/human resources and sourcing decisions e.g. offshoring/outsourcing and personnel changes
Large features spanning over several systems and teams	**(1)** Sharing the same understanding cross multiple teams **(4)** Decision making over multiple teams: **(6)** Coordination and dependency management over multiple / distributed teams **(9)** Interacting with parallel software maintenance (or other organizational processes)
Long timespan	**(1)** Sharing the same understanding cross multiple teams: **(7)** Dealing with changes and unpredictability **(9)** Interacting with parallel software maintenance (or other organizational processes) **(10)** Personnel/human resources and sourcing decisions e.g. offshoring/outsourcing and personnel changes
Complexity of IS architecture and software as a product	**(7)**Dealing with changes and unpredictability
Unpredictable nature of devel-opment process	**(1)** Sharing the same understanding cross multiple teams: (7) Dealing with changes and unpredictability
Other	**(8)**Dealing with large amount of "customers"/stakeholders

Table 5: Problem categories mapped to the features of large scale IS projects.

The challenges recognized in the analysis can be grouped to 10 problem categories. First category (table 4) *"Sharing the same understanding across large organization"* represents the challenge of aligning the mental model over large organization so that shared information is understood similar way. Correct interpretation of information requires having shared common language and culture. In the context of software engineering this means for example common definitions and terminology used in the requirements definition, preferred design principles, coding standards, approval and completion criteria for deliverables, common modelling languages and methods, shared quality criteria and standards and so on. These conventions are easily shared within small team and easily clarified within occasional face to face discussion but within large organization agreeing interpretation each time would cause excessive amount of additional communication and to avoid that distribution of shared conventions over large organization require facilitation.

Recognizing things which need to be shared between teams, how sharing is established and what can be left as internal to team are needed and depend on the organization structure and distribution of work. Shared understanding is especially critical over the deliverables related to features that are split to different teams and their boundaries. Also updating and communicating updated shared models after changes needs to be ensured during long project.

Second category "*Setting roles and responsibilities over large organization*" (table 4) is related to the challenge on defining the optimal organization setting to facilitate all the project dimensions, such as software engineering process selected, software delivery pipeline all the way to delivery to use, business and end user/customer rollout and to enable division and coordination of project content over application domain and functional areas developed. It is common that in a large scale information system project different dimensions may proceed with different pace. E.g. transition to use and to maintenance process may have different process cycle and timing constraints than the implementation, and this needs to be enabled in the project organization. So dividing the large project organization to teams and dividing the work processes within the project (such as software development, enterprise architecture definition and management, release management, testing) across the teams is a challenge that needs to be resolved when setting up a large scale project. Especially setting up parts of organization which execute processes common for all teams, such as acceptance testing, production deployments, trainings etc. may be problematic. The selected methodologies, development processes and how the project scope is defined impacts to the optimal organization.

Third category (in table 4) *"Distributing and assigning tasks for multiple teams"* is related to second category (Setting the roles and responsibilities over large organization). The view point in this category is more about the division of design and development work tasks to teams than about working processes and team boundaries. Distribution of work to teams has a relation to workload and working capacity and hence it will impact the schedules of completing deliverables in the project scope. On the other hand there may be dependencies and constraints in the project application domain that state how the tasks can be assigned to specific teams. Division to functional areas together with technical dependencies may lead to uneven distribution of work and waiting time in some teams.

Category *"Decision making over multiple teams"* (in table 4) groups challenges that are related to making decisions which have wide impact in a large scale organization. Decisions which impact over several project dimensions may not be naturally facilitated by the project working processes. Examples of such things are prioritizations (for example over features split to several development teams), scoping decisions and tools/component selections that impact multiple development teams, negotiations and decisions over architecture (where to implement features that can be resolved multiple ways), configuration changes or exceptional activities in shared environments which may impact all development teams and different levels of testing etc. In hierarchical organization decision making can usually be passed to level in the command chain common to all stakeholders of specific decision but in the flat large scale organization with autonomous teams there may not be a common decision making forum for all necessary participants. Recognizing the impacts of decisions and correct stakeholders and participants to the decision making in any kind of the large scale organization can be difficult and enforcing the decision in cases when there are conflicting goals and interests and no central ownership or authority over the problem is a challenge. Recognizing most common decision cases and facilitation of decision making needs to be designed as part of large scale project set up and it's dependent on the project structure and project processes.

Category five *"Communication over multiple or distributed teams"* (in table 4) is close to first category *"Sharing the same understanding across large organization".* While the first category is about sharing the terminology, conventions and common mental model, the category five is more about ensuring the communication in the first hand. When the organization gets larger, information sharing requires facilitation, tools and processes to reach all necessary receivers. Especially in the case of distributed teams tools and communication media come to important role and processes should ensure using them timely. Information sharing between teams is needed for example in requirements negotiation, when communicating design, test results and incidents, configuration status, progress etc. between different dimensions of project and in knowledge sharing in the transitions from one organizational unit to other. Also communicating deliverables between teams usually requires tools or some kind of media.

Sixth category *"Coordination and dependency management over multiple or distributed teams"* (in table 4) groups multitude of challenges that large scale IS projects have related to management of dependencies and coordination of interactions between teams.

Activities requiring several teams' participation need interaction and coordination between teams. Such activities are for example splitting large end to end features to smaller sub-features and deciding the solution over systems landscape, making architectural decisions impacting widely in the system landscape, recognition and minimization of dependencies and boundaries of resulting such sub-features, designing interfaces/interactions and communicating them, agreeing deliverables over such split features from one team to another and organizing testing of end to end features involving experts of multiple teams.

In addition to shared tasks, synchronization of schedules and monitoring the status of individual teams tasks is often needed. For example completion and delivery of design and implementation deliverables related to split features to the counterpart teams need to be synchronized to avoid delays in other teams work. Also following up completion of sub-features to end to end features for integration, integration testing and end to end testing is needed in order to plan and activate next activity. Following up deliverables of multiple teams from software configuration perspective is needed to keep the configuration status up to date.

Mutual adjustment of schedules and deliverables may be needed in order to keep the software configuration working and to keep the dependencies when managing builds and releases including end to end features. Also aligning testing activities with software configuration status and environments (e.g. what can be tested, what deliveries may be missing, what are the statuses of the applications/systems in the test environment) may be needed especially in the end to end testing of processes and large features.

Dependencies need to be considered and understood when planning timing and meaningful content for releases to end users/customers and overall coordination of the teams towards common milestones and target schedules is needed to reach such targets.

Multiple challenges caused by high amount of changes and unpredictability of software engineering problems in a complex and large scale program are grouped under category seven *"Dealing with changes and unpredictability"* (table 4).

Problem solving process unpredictable nature and software product or IS complexity in a large scale project context added with the long life time of a large scale development project lead to high amount of incomplete requirements and changes to features in all the development process stages and project phases. Requirements stay open long time and changes can occur to finalized requirements, completed design, even completed features and all the way to delivered and accepted features in case project delivery is phased and project is still responsible of these delivered components. This leads to situation where even in the late development or start of testing there may be parts of the requirements incomplete and under investigation, making project phasing difficult. While requirements engineering, design, development and testing are intertwined, requirements engineering activity cannot be ceased before end of project and it is extremely difficult to predict development completion time before everything is delivered and approved. Changes in the requirements mean changes in the individual verification criteria of features and deciding when and how to test and accept features while end results and requirements are not known and fixed until late stage is challenging. Also keeping requirements up to date during long time span of project is needed.

Planning timing and meaningful content for releases while changes to implemented features may already be under development is also challenging. Due to late finalization of requirements release content may not be fixed until nearly release time. It is also difficult to decide release package readiness for approval. While not all conditions can be tested in a large scale contexts due to its complexity, it is difficult to decide how much testing is enough. Readiness of testing depends on the amount and frequency of findings during the testing phase, so time and needed test rounds for set of feature can't be predicted well in advance making scheduling of the approvals more difficult.

While final solution can't be fully defined or possible problem cases predicted all coming activities can’t be recognized in the initiation phase of the project which makes creation of complete plans (schedule estimates, resource needs, recognised work packages and tasks, etc.) in advance impossible. Expected high amount of changes during long lifecycle of the project makes the plans created in the initiation phase even more uncertain the further to the future they reach. Changes during long time span cause lots of re-planning. Unpredictability and inability to model scenarios completely add uncertainty to the decision making in different project activities and in planning and management of the project.

Finally changes in business goals and priorities are also possible during long project lifetime which may impact project feasibility, scope and success. Measuring success of project after lots of changes is challenging.

Challenges in category eight (table 4) *"Dealing with large amount of "customers"/stakeholders"* is recognized outside of the large scale project characteristics collected from the literature.

Large scale IS project typically has large amount of requirements and requirements sources and stakeholders which need to be involved and satisfied. With multiple "customers" the needed interactions and communication and often also time required to set and analyse the requirements and validate and confirm the results is multiplied. Decision making and prioritization become difficult with various stakeholders having contradicting objectives.

Challenges regarding software maintenance in category nine (table 4) *"Interacting with parallel software maintenance (or other organizational processes)"* are associated to long lifetime of and complex end to end features which are characteristic to large scale projects.

In large scale software engineering project it is common that the solution is taken into use during the project while there still are further development and releases coming. Separation between the software maintenance process and the development project may be difficult to define and there may be confusion over what activities are on project responsibility and what on the maintenance organization responsibility, especially regards fixes/patches needed to production software. Interaction between these two streams needs to be carefully planned, e.g. responsibilities over version control, creating, deploying and testing maintenance fixes to project side software branch and controlling changes near new project releases. Also handing over released functionality from development project to maintenance organisation with necessary documentation and trainings may be difficult when changes are expected in near future.

In a large scale IS landscape the maintenance process may be divided to different vendors per applications and it is common that the different support lines may involve different organizations or vendors. Agreeing and setting up maintenance in such large scale environment is big effort in itself. Especially defining and agreeing incident management process and responsibilities over large end to end processes or features involving several applications and therefore several maintenance organization may be complex task.

Finally category ten (table 4) *"Personnel/human resources and sourcing decisions e.g. offshoring/outsourcing and personnel changes"* includes challenges in the human resources area. Large scale IS project involves lots of personnel and human resources and sourcing issues are have major impact on the project performance. Personnel offshoring/outsourcing are often considered and (customer) organization policies which are outside of the project may impact on the decisions. Personnel changes are also likely during long project and introducing new people to project require special attention (such as trainings) and learning time before performing fully. Also group dynamics aspects impact on project performance.

4 Agile Practises Response to Challenges of Large Scale

Following chapters present the agile practises that represent the agile methodologies in the analysis, the analysis of agile practises impact on the challenges related to large scale software development or IS projects as presented in chapter 3, the results of the analysis and the discussion of the results.

4.1 Agile Practises

Agile methodologies differ from each other in the process details, they have both similar elements and differences. To get results how agile methodologies as an overall group of methodology practise respond to challenges of large scale IS projects, analysis needs to be done in more granular level than methodologies. Three possible sources for the analysis was recognised during the literature study for chapters 2 and 3.

Agile Alliance's Guide to Agile defines 60 agile practices at the time of this analysis [GtA15]. Practices described in the Agile Alliance site are in different levels, many of them very atomic and variations of same practise are listed as separate practises. E.g. "Three Questions" used in the daily meeting are one practice while the "Daily Meeting" itself is one practise.

State of the agile survey [SoA16] on the other hand defines 25 agile techniques which correspond to practises recognized by Agile Alliance. State of the agile is not a scientific resource and the summary report does not give explanation of the origins of techniques in State of agile –survey. It is also possible that techniques not reported as used by respondents may have been omitted from the survey.

Agile practises have been recognised also in the scientific research literature. Diebold and Dahlem have listed unique 18 agile practises in their mapping study regarding agile practices usage [DiD14]. Their study gathers practises from different agile methodologies under common nominators but the study does not include explanations to the named practices.

Practises from Agile Alliance were chosen to basis for this analysis mainly because the original descriptions of each practise are available and there is no risk of misunderstanding or wrong interpretations of the practises. While Agile Alliance is not a scientific source it is anyhow global organization representing a world-wide community of agile practitioners committed to advancing Agile development principles and practices and can hence be considered as standard source of agile practises in use.

4.2 Analysis of Agile Practises Impact to issues of Large Scale IS Projects

Each listed agile practise was considered against each of the 10 challenges resulting from the analysis in previous chapters to determine whether the practise has a mitigating impact on the particular challenge. This was done by reviewing the description of each practise in order to detect any impact regarding the challenges in the specific problem category. For example following citations can be found of the definition of the first practise “Acceptance Testing” [GtA15]: *“An acceptance test is a formal description of the behaviour of a software product…”* and *“For many Agile teams acceptance tests are the main form of functional specification…”* Also in the benefits section of the description it is mentioned that acceptance testing is *“…encouraging closer collaboration between developers on the one hand and customers, users or domain experts on the other, as they entail that business requirements should be expressed…”* and *“...providing a clear and unambiguous "contract" between customers and developers…”* Based on these statements acceptance testing is considered impacting the problem categories 1 and 5 related to shared understanding and communications. Result of analysis is presented in the table 6.

Agile practises	Problem categories of large sale software engineering or IS project										*count*
	Sharing the same understanding across large organization	Setting roles and responsibilities over multiple teams	Distributing and assigning tasks for multiple teams	Decision making over multiple teams	Communication over multiple / distributed teams	Coordination over multiple / distributed teams	Dealing with changes and unpredictability	Dealing with large amount of "customers"/stakeholders	Interacting with parallel software maintenance (or other organizational processes	Personnel/HR and sourcing decisions	
Acceptance testing	P	O	O	O	P	O	O	O	O	O	*2*
ATDD	P	O	O	O	P	O	O	O	O	O	*2*
Automated build	O	O	O	O	O	O	O	O	O	O	*0*
Backlog grooming	O	O	O	O	O	O	P	O	O	O	*1*
Backlog	P	O	P	O	P	O	P	O	O	O	*4*
BDD	P	P	O	O	P	O	O	O	P	O	*4*
Burndown chart	O	O	O	O	P	O	O	O	O	O	*1*
Collective ownership	O	O	O	O	O	O	O	O	O	O	*0*
Continuous deployment	O	O	O	O	O	O	O	O	O	O	*0*
Continuous integration	O	O	O	O	O	O	O	O	O	O	*0*
CRC cards	O	O	O	O	O	O	O	O	O	O	*0*
Daily meeting	O	O	O	O	O	O	O	O	O	O	*0*
Definition of Done	O	P	O	O	P	O	O	O	O	O	*2*
Definition of Ready	O	P	O	O	O	O	O	O	O	O	*1*
Estimation	O	O	O	O	O	O	O	O	O	O	*0*
Exploratory testing	O	O	O	O	O	O	O	O	O	O	*0*
Facilitation	O	O	O	O	O	O	O	O	O	O	*0*
Frequent releases	O	O	O	O	O	O	P	P	O	O	*2*
Given - When - Then	P	O	O	O	O	O	O	O	O	O	*1*
Heartbeat retrospective	O	O	O	O	O	O	O	O	O	O	*0*

Agile practises	Problem categories of large sale software engineering or IS project										count
	Sharing the same under-standing across large or-ganization	Setting roles and responsi-bilities over multi-ple teams	Distrib-uting and assigning tasks for multiple teams	Decision making over multi-ple teams	Communi-cation over multiple / distributed teams	Coordina-tion over multiple / distributed teams	Dealing with changes and unpre-dictability	Dealing with large amount of "custom-ers"/stake-holders	Interacting with paral-lel software mainte-nance (or other or-ganiza-tional pro-cesses	Person-nel/HR and sourcing decisions	
Incremental devel-opment	O	O	O	O	O	O	**P**	O	O	O	*1*
Information radia-tors	O	O	O	O	**P**	O	O	O	O	O	*1*
Integration	O	O	O	O	O	O	O	O	O	O	*0*
Invest	**P**	O	O	O	O	O	O	O	O	O	*1*
Iteration	O	O	O	O	**P**	O	O	O	O	O	*1*
Iterative develop-ment	O	O	O	O	O	O	**P**	O	O	O	*1*
Kanban board	O	O	O	O	**P**	O	O	O	O	O	*1*
Lead time	O	O	O	O	O	O	O	O	O	O	*0*
Milestone retro-spective	O	O	O	O	O	O	O	O	O	O	*0*
Mock objects	**P**	O	O	O	O	**P**	O	O	O	O	*2*
Niko-niko calendar	O	O	O	O	O	O	O	O	O	O	*0*
Pair programming	O	O	O	O	O	O	O	O	O	O	*0*
Personas	O	O	O	O	O	O	O	O	O	O	*0*
Points (estimates in)	O	O	O	O	O	O	O	O	O	O	*0*
Planning poker	O	O	O	O	O	O	O	O	O	O	*0*
Project chartering	**P**	O	O	O	O	O	O	O	O	O	*1*
Quick design ses-sion	O	O	O	O	O	O	O	O	O	O	*0*
Refactoring	O	O	O	O	O	O	O	O	O	O	*0*
Relative estimation	O	O	O	O	O	O	O	O	O	O	*0*
Role-feature-reason	**P**	O	O	O	O	O	O	O	O	O	*1*

Agile practises	Problem categories of large sale software engineering or IS project										count
	Sharing the same under-standing across large or-ganization	**Setting roles and responsi-bilities over multi-ple teams**	**Distrib-uting and assigning tasks for multiple teams**	**Decision making over multi-ple teams**	**Communi-cation over multiple / distributed teams**	**Coordina-tion over multiple / distributed teams**	**Dealing with changes and unpre-dictability**	**Dealing with large amount of "custom-ers"/stake-holders**	**Interacting with paral-lel software mainte-nance (or other or-ganiza-tional pro-cesses**	**Person-nel/HR and sourcing decisions**	
Rules of simplicity	O	O	O	O	O	O	O	O	O	O	*0*
Scrum of Scrums	**P**	**P**	**P**	**P**	**P**	**P**	O	O	O	O	*6*
Sign up for tasks	O	O	**P**	O	O	O	O	O	O	O	*1*
Simple design	O	O	O	O	O	O	**P**	O	O	O	*1*
Story splitting	O	O	O	O	O	O	O	O	O	O	*0*
Story mapping	O	O	O	O	O	**P**	O	O	O	O	*1*
Sustainable pace	O	O	O	O	O	O	O	O	O	O	*0*
Task board	O	O	O	O	**P**	O	O	O	O	O	*1*
TDD	O	O	O	O	O	O	O	O	O	O	*0*
Team	O	**P**	O	O	O	O	O	O	O	O	*1*
Team room	O	O	O	O	O	O	O	O	O	O	*0*
Three C's	O	O	O	O	O	O	O	O	O	O	*0*
Three questions	O	O	O	O	O	O	O	O	O	O	*0*
Timebox	O	O	O	O	**P**	O	O	O	O	O	*1*
Ubiquitous lan-guage	O	O	O	O	O	O	O	O	O	O	*0*
Unit testing	O	O	O	O	O	O	O	O	O	O	*0*
Usability testing	O	O	O	O	O	O	O	O	O	O	*0*
User stories	**P**	O	O	O	**P**	O	O	O	O	O	*2*
Velocity	O	O	O	O	O	O	O	O	O	O	*0*
Version control	O	O	O	O	O	O	O	O	O	O	*0*
Count	*11*	*5*	*3*	*1*	*13*	*3*	*6*	*1*	*1*	*0*	

Table 6: Mapping of agile practises impacting challenges of large scale projects (P=has partial impact, O=has no impact).

4.3 Analysis Results of Agile Practises Impact to issues of Large Scale IS Projects

Challenge groups that were impacted most based on the analysis were: *5) "Communication over multiple or distributed teams"* (impacted by 13 of the 60 practises), *1) "Sharing the same understanding across large organization"* (11 of the 60 practises) and *7) "Dealing with changes and unpredictability"* (6 of the 60 practises).

Practises that were least impacted were *4) "Decision making over multiple teams"* (1 practise of 60), *8) "Dealing with large amount of "customers"/stakeholders"* (1 of 60), 9) *"Interacting with parallel software maintenance (or other organizational processes)"* (1 of 60) and 10) *"Personnel/human resources and sourcing decisions e.g. offshoring/outsourcing and personnel changes"* (no impacting practises recognised).

As the challenge categories *5) "Communication over multiple or distributed teams"* and *1) "Sharing the same understanding across large organization"* were related to each other, they are also mitigated with some of the same practices. Acceptance Tests used in "Acceptance Testing" and in "ATDD" (Acceptance Test Driven Development) can be considered a form of documentation of the requirement. Formal documentation assists communication also between multiple and distributed teams. "BDD" (Behaviour Driven Development) defines the notation used between developers, domain experts and testers and facilitates the communication between the mentioned roles. Tools designed for BDD usage usually include features to automatically create documentation of the designed features. Both ADD and BDD also contribute on sharing the same understanding over the criteria of completed deliverables related to requirements definition or development. These practises are typically used within members of the same team, and not directly designed to be used between teams, but the deliverables and conventions are valid also in inter-team communication. Common "Backlog", if it is shared between all teams, works as a single source defining the work to be done in a form of "User Stories" which both serves the communication and sharing understanding of the project scope. Although a decision needs to be made whether to have one common backlog for all teams or team internal backlog? "Scrum of Scrums" can be used as a both communication forum and final forum where mismatches of understanding recognized during the project activities can be raised and alignment actions made, it is the only actual practise intended to address cross team issues.

Practises that are designed to communicate things inside the team can also contribute to communications between teams mitigating the challenges in category *5) "Communication over multiple or distributed teams"*. For example "Burndown Charts" can be used to communicate status also to other teams, if they are clarified with the shared "Definition of Done" over teams and used in a context of "Iteration" or "Time-Box" known or common to other teams. "Information Radiators" may communicate information to other teams, more so if they are made available over digital media, but they are primarily targeted for internal purposes and nothing ensures information is received by others (information radiators can either be seen only by accident or other teams must intentionally seek the information). Also "Kanban (board)" or "Task Board" may communicate information of team internal status and scope to other teams as well, but like Information Radiators, they are intended primarily as team internal tool.

Practises that set common conventions over requirements or acceptance tests definition also contribute on sharing the same understanding, language and terminology over multiple or distributed teams therefore impacting challenges in *category 1) "Sharing the same understanding across large organization"*. Usage of common templates and formulas "Given-When-Then", "Role-Feature-Reason" and "Invest" are such practises. Furthermore "Mocks" can also be used as form of agreement and definition over interfaces if developed and kept up to date as per common agreement. If "Project Charter" is commonly created by and shared over all teams it can unify the understanding of project goals, though Project Charter is intended to be "known and approved by all members of the team" and is by definition internal to team. Even though team internal Project Charter can be information radiator visible to other teams and therefore may contribute on sharing the same understanding.

Total of six practises was recognized mitigating the challenges in category *7) "Dealing with changes and unpredictability"*. Even though the amount was less than for the two previous most impacted categories, the mitigating impact of these practises is clearer. This is mainly because challenges in this category are not related to the organization but more for the large application domain scope and long lifetime of large scale IS project.

Using "Backlog" as evolving and hierarchical specification gives a tool to manage the project scope, priorities and scope changes. With "Backlog Grooming" technique scope, goals and priorities can be kept up to date and so called scope creep prevented even while changes occur. "Frequent Releases" together with "Incremental Development" gives mechanism to deal with changes and unpredictability; while planning only short term and expecting feedback before planning next release, re-planning far in the future is avoided. Frequent Releases also prevent scope creep and changes and problems accumulating and causing delay that is revealed only late in the project schedule. Incorporating changes to already developed content is enabled with "Iterative development". Finally "Simple Design" principle supports incremental development and responses to unpredictability by aiming for avoiding unnecessary costs of preparing for something that is not needed after all.

Five practises were recognized having some impact over challenges in category *2)"Setting the roles and responsibilities over large organization,* two of them more directly addressing the question of organizational setting of large organization and three impacting the role definitions within the software development process.

Practise of "Team" directs the organization set up to construct teams of all necessary technical (programming, designing, testing) or business (domain knowledge, decision making ability) competencies. "Scrum of Scrums" guides the organization setting to divide the large groups into agile teams of 5-10 and to have additional daily meetings with ambassadors of all teams. In addition to these organizational practises "Definition of Done" and "Definition of Ready" both communicate the limits of the role and responsibilities of the person to whom the task is assigned to before reaching the status ready or done in task lifecycle, helping define the boundaries of e.g. requirements definition responsibilities and developer role. Similarly "BDD" (Behaviour Driven Development" guides the conversation between developers, domain experts and testers. These practises doesn't consider other roles than directly development oriented, such as how to organize roles related to trainings, rollout, release and deployment management for example or guidance over competence area across teams (e.g. architecture decisions). The latter practises do not actually guide the organizational structure and set up to teams but they clarify the role boundaries internal to teams if teams are set according to the "Team" practise.

For category *3) "Distributing and assigning tasks for multiple teams",* only three practises where recognised which to some extent address how the tasks are assigned between teams. "Backlog" can be used to define only tasks assigned to team in which case mechanism is needed to decide the division to backlogs. Agile Alliance does not define a practise for this. Common way to do this is the division to product areas (not included in the practises) or per applications in an enterprise application domain, but this does not ensure even work distribution over teams. Practise “Sign up for task” means that individual team members can then choose a task from the team backlog to work with. It is also possible to have shared “Backlog” and in that case “Sign up for task” would mean individual assigning the task to himself and to the team he belongs to at the same time. In that case the only forum to negotiate these choices between other teams would be “Scrum of scrums”.

Challenges in category *6) “Coordination and dependency management over multiple or distributed teams”* were mitigated with three practises also. “Scrum of scrums” is only actual coordination forum for dependency management listed in the practises. Using “Mock Objects” in the development and system testing can hide the dependencies during development and unit/system testing time, but for integration, acceptance testing and release of end to end features real objects are needed. Anyhow responsibilities over creation of Mock Objects need to be agreed, the definition and creation of objects coordinated as well as the changes that occur during the time, which in turn adds the need for interaction and coordination before and during the development. So the decoupling impact of this practise regarding the dependencies is only temporary and does not remove the need of coordination over multiple teams although it changes the timing of the needed coordination. “Story mapping” technique may be useful on the recognition of the dependencies, even though the real intent is to help designing feature increments. Management of technical dependencies is not possible without support of working “Version Control”, but Version Control alone does not solve the dependency management problem. This was not considered as mitigating practise, since it is more a requirement than enabler. Possibility of using practise “Collective (code) Ownership” to mitigate dependency management challenges was also considered during the analysis. With Collective Ownership defined so that all teams can change all components in the system landscape, such dependencies where multiple teams contribute to same end to end feature could be avoided. On the other hand, this kind of collective ownership will not remove physical dependencies and need to synchronize the deployment, testing and release schedules of these components. Without having those coordinated as well, collective code ownership over whole landscape would not be possible. Such setting would also require large set of different skills (needed to develop any system in the landscape) from all teams, which is uncommon. Since such practises are not defined, Collective Ownership was not considered as mitigating practise for coordination and dependency management.

The only facilitating practise for making decisions over teams is “Scrum of scrums”, so this is the only practise directly mitigating challenges in category *4) "Decision making over multiple teams"*. Shared "Project Charter" could unify priorities over teams easing decision making in a conflict situation, but as stated before, Project Charter is intended to be internal to team and creating shared Project Charter would require additional collaboration mechanisms over teams. The activity of creating a shared Project Charter as intended in this practice could be very difficult task in itself since it would need to involve the whole project organization. In addition, project charter only guides in the decision making, but does not really facilitate it and not all decisions are directly related to content of project charter. Project Chartering was not counted as mitigating practise for decision making over teams since it is defined to be team internal activity.

Only one practise could be considered mitigating challenges in category *8) "Dealing with large amount of "customers"/stakeholders"*. Using practise of "Frequent Releases" makes it possible to demonstrate value and get feedback from customers early. This strategy doesn't have impact on situations multiple customers having conflicting objectives and priorities but it may raise these situations into awareness more quickly. Frequent releases can also include beta releases to targeted user groups.

Also category *9) "Interacting with parallel software maintenance (or other organizational processes)"* is impacted only one practise. If "BDD" (Behaviour Driven Development) tools are used, they usually offer also automated creation of end user documentation which is useful also in the transition situation.

No practise was recognised directly impacting the category *10) "Personnel/human resources and sourcing decisions e.g. offshoring/outsourcing and personnel changes"*. Although all communication and all shared understanding may speed up learning of new personnel during the project therefore having mediated impact. Usage of "Collective Code Ownership" practise usually implies that code is well documented and easy to comprehend which also helps new developers, in addition support from other team members may be easier to get when everyone has responsibility over the code.

The single practise that had the most impact on the challenges was Scrum of Scrums. It was recognised to have mitigating impact on 6 of the 10 challenges. Scrum of Scrums is the only practise designed on scaling agile methodologies to larger contexts.

From the analysis it can be found that 33 of the 60 listed agile practises have no mitigating impact to the challenges related large scale IS project. These practises were Automated

Build, Collective Code Ownership, Continuous Deployment, Continuous Integration, CRC cards, Daily Meetings, Estimation, Exploratory testing, Facilitation, Heart Beat Retrospective, Integration, Lead time, Milestone Retrospective, Niko-Niko Calendar. Pair Programming, Personas, Points (estimates in), Planning Poker, Quick Design Sessions, Refactoring, Relative Estimations, Rules of Simplicity, Story Splitting, Sustainable Pace, TDD, Team Room, Three C's, Three Questions, Ubiquitous Language, Unit Testing, Usability Testing, Velocity and Version Control

4.4 Summary of Analysis Results of Agile Practises Impact

No practises were recognised aiming to communication between teams except Scrum of Scrums. Several practises were found to facilitate team internal communication which can by accident also aid the external communication. Found communication related practises cover only software development from requirements to testing, not the cross-functional project dimensions. Development related conventions can unify the understanding and hence also ease communication if they are common for all project across the teams.

Six practises recognised related to coping with frequent changes and unpredictability are independent of the organization size and therefore suitable also in large scale context.

Only two practises were found to guide the organizational setting. These practises do not address the cross functional project dimensions, but are concentrated to the software development aspect. In addition to these, three practises were recognized impacting role boundaries within the software development (requirements definition to testing) dimension.

Backlog and sign up for tasks are the only practises related to distributing and assigning tasks. These practises do not define how to actually divide the work to teams but consider about individual team member aspect of the task assignment. Scaling these practises in a multi-team context is not defined in the practises and requires adapting the practises.

Only practise recognised to facilitate coordination over multiple teams was Scrum of Scrums, in addition two practises was recognised related to dependency management, the other for temporarily loosening the dependencies and the other for dependency recognition.

Scrum of scrums was also found to be the only practise facilitating decision making over multiple teams. One practise (frequent releases) was found facilitating having large amount of customers or stakeholders and similarly only one practise was recognized impacting positively in the situation where software maintenance is working in parallel with the project. No practises was found to facilitate resolving conflict situations with multiple customers having contradicting priorities. Also no practises was found related to personnel management or sourcing decisions.

5 Discussion

Three research questions were set to define the study goal and guide the analysis.

I. What are the characteristics specific to large scale software engineering or Information Systems project?

II. What are the challenges caused by these characteristics?

III. How agile methodologies mitigate these challenges?

To answer the research question ***I)*** existing research literature was investigated and characteristics of large scale IS projects were collected. Six characteristics were recognized; *Multiple teams, Distributed teams, Large features spanning over several systems and teams, Long timespan, Complexity of IS architecture and software as a product* and *Unpredictable nature of development process*.

For research question ***II)*** challenges associated to characteristics of large scale software development or IS projects where first recognised from the literature and then complemented with the analysis against SWEBOK knowledge areas. Analysis resulted ten problem categories; *1) Sharing the same understanding across large organization, 2) Setting roles and responsibilities over multiple teams, 3) Distributing and assigning tasks for multiple teams, 4) Decision making over multiple teams, 5) Communication over multiple/distributed teams, 6) Coordination and dependency management over multiple/distributed teams, 7) Dealing with changes and unpredictability, 8) Dealing with large amount of "customers"/stakeholders, 9) Interacting with parallel software maintenance (or other organizational processes)* and *10) Personnel/human resources and sourcing decisions e.g. offshoring/outsourcing and personnel changes*.

Challenges found are aligned with research challenges 1-5 and 7 suggested as a result of International Conference on Agile Software Development on year 2013. These research challenges were “Inter-team coordination”, “Large project organization / portfolio management”, “Release planning and architecture”, “Scaling agile practices”, “Customer collaboration” and “Knowledge sharing and improvement”. In addition, suggested research agenda included two other topics: “Large-scale agile transformation” and “Agile contracts” [DiM13]. Revised research agenda from International Conference on Agile Software Development on year 2014 included also similar slightly refined five topics matching to the challenges presented in this study; “Organisation of large development efforts”, “Inter-team coordination”, “Knowledge sharing and improvement”, “Release planning and architecture”, “Customer collaboration” and “ Scaling agile practices”. In addition revised research agenda suggests five topics not present in the challenge categories listed in this study; “Agile contracts”, “Agile transformation”, “UX design”, “Key performance indicators in large development efforts” and “Variability factors in scaling“[DiM14].

To address research question ***III)*** agile practises defined by Agile Alliance were selected to represent overall group of agile methodology practise instead of considering each method separately. Each agile practise was considered against each problem category in order to decide whether it has mitigating impact on the issues within category. Results of the research question *III)* and conclusions are presented in the following subchapter.

5.1 Conclusions

As a result of the analysis of agile practises mitigating impact on the challenges of large scale IS projects it was found that:

Changes and unpredictability are directly addressed by 6 of 60 practises promoted by Agile Alliance. Practises facilitating communication and shared understanding were well present, but even though it was not always directly stated, it was clear from the definition and considering the context (and co-existence with other practises) that recognised practises were mainly designed to be utilized within team. 13 practises which could have positive impact also to communication challenges between teams if utilized in certain way were recognized out of 60, similarly 11 practises could possibly impact also to shared understanding between teams. These results are aligned with agile principles “Individuals and interactions over processes and tools” and “Responding to change over following a plan”

Only 5 out of 60 practises were recognized having partial impact on setting the roles and responsibilities within software development process. 3 practises were found having impact on distributing and assigning tasks, of which 1 related to the negotiations over tasks and 2 were on team member level but possibly scalable over teams by adapting practises. 3 practises were related to coordination over multiple teams, 1 of them directly related to coordination and 2 related to dependency recognition and removal of technical dependencies that would require coordination. In addition only 1 practise was found having partial impact on issues related to decision making over multiple teams, 1 on large amount of stakeholders and 1 on parallel software maintenance. No practises facilitating conflict resolving in situations with multiple customers having contradicting priorities was found and no practises were recognised related to personnel and sourcing issues in large scale projects.

All practises were defined in individual team member level or as team internal practises.

Practises were considered having only partial or moderated impact as such, or possibly having impact if adapted and scaled to be utilized as inter-team practises.

Only practise designed to scale agile methodologies over larger organizational setting is Scrum of Scrums. No other practises intended to facilitate collaboration between teams was found. All practises are primarily targeted to facilitate work within team.

All found practises were targeting only software development process roles and activities.

Presented practises do not address the cross functional project dimensions (trainings, release planning, deployments and rollouts of business functionalities, transitions to maintenance organizations etc. requiring interaction with the software development pipeline, such as knowledge transfers, environment set ups, deployments, fixing the late bugs).

Based on the analysis agile practises will benefit large scale software development and IS projects in the team level by enhancing the team level performance and in mitigating the challenge of dealing with changes and unpredictability. Challenges related to large scale project context still remain cross teams and overall project level.

Following needs for adaptation, alignment over teams and additional practises were recognised from the analysis results. Results are also summarised in the table 7.

Adaptations of practises related to distribution, assignment and follow up of tasks e.g. Backlog and Sign up for tasks are needed in order to scale the practises to be used over large scale project.

Practises related to software development process, ways of working and common principles should be aligned over teams. For example testing related practises (to some extent) ATDD, BDD, Acceptance Tests, requirements definition related practises such as User Stories, Given-When-Then, Role-Feature-Reason and Invest, process boundaries related practises like Definition of Done and Definition of Ready and timing related practises like Iteration and Time-Box. Also a techniques used in dependency management like Story Mapping and Mocks need alignment over teams.

If team internal mechanisms are used also for external communication additional aligned practises are needed in order to publish and make the information available, this could be considered for example Burndown Charts, Information Radiators, Kanban (boards), Task Boards and Project Charters.

Additional practises are needed to facilitate collaboration between teams, address interactions with the cross functional project dimensions and strengthen the dependency management and decision making. Single Scrum of Scrums meeting is not enough to cover large scale project cross team coordination needs, so some adaptation or additions are likely to Scrum of Scrums practise as well. Also practises to manage large amount of "customers"/stakeholders and personnel/human resources and sourcing issues need to be considered.

Problem category	**Practises directly mitigating the challenges**	**Practises which impact but need adaptation to scale**	**Practises which need to be aligned over teams**	**Additional practises especially needed to mitigate challenges**
(1) Sharing the same understanding across large organization	- Scrum of Scrums	- Backlog	- Acceptance testing - ATDD - BDD - Given - When - Then - Invest - Mock objects - Project chartering - Role-feature-reason - User stories	-
(2) Setting roles and responsibilities over multiple teams	- Scrum of Scrums - Team	-	- BDD - Definition of Done - Definition of Ready	-
(3) Distributing and assigning tasks for multiple teams	- Scrum of Scrums	- Backlog - Sign up for tasks	-	-
(4) Decision making over multiple teams	- Scrum of Scrums	-	-	- Additional practices needed
(5) Communication over multiple / distributed teams	- Scrum of Scrums	- Backlog	- Acceptance testing - ATDD - BDD - Burndown chart - Definition of Done - Information	- Additional practises needed for communicating with other project dimensions than direct software

Problem category	**Practises directly mitigating the challenges**	**Practises which impact but need adaptation to scale**	**Practises which need to be aligned over teams**	**Additional practises especially needed to mitigate challenges**
			radiators - Iteration - Kanban board - Task board - Time-box - User stories	engineering
(6) Coordination and dependency management over multiple / distributed teams	- Scrum of Scrums	-	- Mock objects - Story mapping	- Additional practices needed for collaboration between teams, and strengthening the dependency management
(7) Dealing with changes and unpredictability	- Backlog - Backlog grooming - Frequent releases - Incremental development - Iterative development - Simple design	-	-	-
(8) Dealing with large amount of "customers"/ stakeholders	- Frequent releases	-	-	- Additional practises to be considered
(9) Interacting with parallel software maintenance (or other organizational processes)	-	-	- BDD	- Additional practises to be considered
(10) Personnel/human resources and sourcing decisions e.g. offshoring/outsourcing and personnel changes	-	-	-	- Additional practises to be considered

Table 7: Agile practises which mitigate the challenges of large scale IS projects and adaptation and addition needs.

5.2 *Validity and future work*

Source material for features and challenges of large scale IS projects consisted solely of research papers having agile project context. It is possible that different challenges would have been recognized from the research regarding non agile projects. It is also recognized that methodology itself moderates the impact of success factors. According to some studies, contingency fit or misfit between methodology and project conditions impacts on which success factors have significance. Therefore different success factors have different impact on the project success depending on the methodology approach [ACD15]. To get more generalizable results this study could be amended with additional analysis from plan based projects or with systematic literature review.

Since the analysis of agile practises impact to challenges of large scale IS projects was done as a theory level table study, it is recommendable to continue with verifying these results with case studies of existing projects, concentrating on used agile practises and their impact, additional scaling mechanisms developed, found challenges related to large scale characteristics and how they were mitigated in the projects.

According to State of Agile Survey, three most used scaling mechanisms are Scrum of Scrums, SAFe (Scaled Agile Framework) and company internally created methods [SoA16]. The analysis of agile methodologies in large scale projects contexts is suggested to be continued with analysis of SAFe, possibly also other scaling mechanisms and whether those addresses the challenges recognized in the study. Scaled Agile Framework is a framework for scaling agile development over large development organization and it includes practises targeted to team level, program level and portfolio level. Large development organizations are not the same as large scale development projects, but there are similarities and therefor some of the program and portfolio level practises could be applied and benefitting large scale projects as well [Laa14, Lef11].

6 Summary

This study investigated benefits and challenges of agile methodologies on the large scale software development and information systems projects by recognizing the features of large scale projects, analysing the challenges related to them from existing research literature and using SWEBOK knowledge areas and by analysing the impact of agile practises listed by Agile Alliance to the recognized challenges.

As a result it was recognized that while the agile practises enhance the team level performance and provide direct practises to manage challenges regarding high amount of changes and unpredictability of problem solving process of a large scale IS project challenges still remain on the cross team and overall project level.

Conclusion from the analysis is that large scale software development and IS projects benefit from using agile methodologies. However when seeking best fit between methodology and project characteristics or model where agile approach would respond to the characteristics of the large scale project context which would likely contribute to project success, both adaptations of current practises and developing additional practises are needed.

Following areas for adaptations and new practises are suggested for scaling agile methodologies over large scale project contexts based on the analysis.

1) Adaptation of practises related to distribution, assignment and follow up of tasks in order to scale them over multiple teams of large scale project.

2) Alignment of practises related to software development process, ways of working and common principles over all teams.

3) Developing additional practises to facilitate collaboration between teams, to ensure interactions with the cross functional project dimensions and to strengthen the dependency management and decision making between all project dimensions such as mentioned in chapter 1 regarding IS systems projects dimensions additional to software engineering.

4) Possibly developing and aligning practises to facilitate teams' external communication, such as publish status or other relevant information all teams.

The study produced comprehensive explanation of the extent and manifestation of challenges related to large scale software development and IS project characteristics and detailed impact of agile practises to these challenges. This information and the suggested areas for adaptation and additional practises should prove to be useful for software development and IS project practitioners when considering agile method adoptions or adaptations in a large scale project context.

References

ACD15	Ahimbisibwe, A., Cavana, R.Y., Daellenbach, U., A contingency fit model of critical success factors for software development projects: A comparison of agile and traditional plan-based methodologies. Journal of Enterprise Information Management, 28,1(2015), p.7-33.
ASR02	Abrahamsson P., Salo O., Ronkainen J., Warsta J., Agile software development methods, Review and analysis. VTT Technical Research Center of Finland, Helsinki, Finland, 2002.
BTB03	Boehm B., Turner R., Booch G., Cockburn A., Pyster A., Balancing Agility and Discipline: A Guide for the Perplexed 1st Edition. Addison-Wesley/Pearson Education, 2003.
ChC08	Chow, T., Cao, D.-B., A survey study of critical success factors in agile software projects. Journal of Systems and Software, 81,6(6/2008), p.961-971.
DFI14	Dingsøyr T., Fægri T.E., Itkonen J., What Is Large in Large-Scale? A Taxonomy of Scale for Agile Software Development. Proceedings of the 15th International Conference on Product-Focused Software Process Improvement, PROFES 2014, Helsinki, Finland, December 10-12, 2014, p.273-276.
DiD14	Diebold, P., Dahlem, M., Agile practices in practice - A mapping study. Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, ACM New York, NY, USA, 2014, article 30.
DiM13	Dingsøyr T., Moe N.B., Research Challenges in Large-Scale Agile Software Development. In: ACM SIGSOFT Software Engineering Notes, 38,5(9/2013), p.38-39.
DiM14	Dingsøyr, T., Moe, N.B., Towards Principles of Large-Scale Agile Development: A Summary of the workshop at XP2014 and a revised research agenda. In: Lecture Notes in Business Information Processing, 199 (5/2014), p.1-8.
DyD08	Dybå, T., Dingsøyr, T., Empirical studies of agile software development: A systematic review. International Journal of Information and Software Technology, 50,9-10(9/2008), p.833-859.
DyD15	Dybå, T., Dingsøyr, T., Agile Project Management: From Self-Managed Teams to Large-Scale Development. Proceedings of the 37th International Conference on Software Engineering (ICSE), IEEE/ACM, Florence, Italy, May 16-24, 2015, vol.2, p.945-946
GBT15	Gregory, P., Barroca, L., Taylor K., Salah D., Sharp H., Agile Challenges in Practice: A Thematic Analysis. Proceedings on 16th International Conference on Agile Processes in Software Engineering and Extreme Programming, XP 2015, Helsinki, Finland, May 25-29, 2015, p.64-80.
GtA15	Guide to Agile. Agile Alliance, 2015. https://www.agilealliance.org/agile101/guide-to-agile/. [24/4/2016]

GuD15	Gupta, D., Dwivedi, R., A framework to support evaluation of project in-hand and selection of software development method. Journal of Theoretical and Applied Information Technology, 73,1(3/2015), p.137-148.
ICB06	ICB - IPMA Competence Baseline, Version 3.0. International Project Management Association, 2006. . [Also http://www.ipma.world/about/, 9.5.2016]
KeL05	Kettunen, P., Laanti, M., How to steer an embedded software project: tactics for selecting the software process model. International Journal of Information and Software Technology, 47,9(6/2005), p.587-608.
Laa14	Laanti M., Characteristics and Principles of Scaled Agile. In: Lecture Notes in Business Information Processing, 199 (5/2014), p.9-20.
LaV09	Larman C., Vodde B., Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum 1st Edition. Addison-Wesley Professional, 2008.
Lef11	Leffingwell, D., Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise. Addison-Wesley, 2011.
MSP09	Managing Successful Projects with PRINCE2®, 2009 Edition. AXELOS, 2009. [Also https://www.axelos.com/best-practice-solutions/prince2, 8/5/2016]
Pap14	Papadopoulos G., Moving from traditional to agile software development methodologies also on large, distributed projects. Proceedings of the 3rd International Conference on Strategic Innovative Marketing (IC-SIM 2014), Madrid, Spain, Sept 1-4, 2014, p.455-463.
PMB13	PMBOK® Guide, A Guide to the Project Management Body of Knowledge Fifth Edition. Project Management Institute, 2013. [Also http://www.pmi.org/default.aspx, 8/5/2016]
RaA14	Razavi A.M., Ahmad R., Agile Development in Large and Distributed Environments: A Systematic Literature Review on Organizational, Managerial and Cultural Aspects. Proceedings of 8th Malaysian Software Engineering Conference (MySEC), Langkawi, Malaysia, Sept 23-24, 2014, p.216-221.
SAR12	Savolainen P., Ahonen JJ., Richardson I., Software development project success and failure from the supplier's perspective: A systematic literature review. International Journal of Project Management 30,4(5/2012), p.458-469.
SeP15	Serrador P., Pinto J.K., Does Agile work: A Quantitative analysis on agile project success. International Journal of Project Management, 33,5(7/2015), p.1040–1051.
SEP13	Software Extension to the PMBOK® Guide Fifth Edition. Project Management Institute, 2013.
SHK14	Scheerer A., Hildenbrand T., Kude T., Coordination In Large-Scale Agile Software Development: A Multiteam Systems Perspective. Proceedings of 47th Hawaii International Conference on System Sciences, Waikoloa, HI, Jan 6-9, 2014, p.4780-4788.

SoA16	10th Annual State of the Agile Survey. Version One, 2016. [Also http://stateofagile.versionone.com/, 6/4/2016]
SWE14	SWEBOK V3.0 Guide to Software Engineering Body of Knoweldge. IEEE Computer Society, 2014. [Also www.swebok.org, 14/3/2016]
TRA15	Tripathi N., Rodríguez P., Ahmad M.O., Oivo M., Scaling Kanban for Software Development in a Multisite Organization: Challenges and Potential Solutions. Proceedings of 16th International Conference on Agile Software Development, XP 2015, Helsinki, Finland, May 25-29, 2015, p.178-190.
VlV15	Vlietland, J., Van Vliet, H., Towards a governance framework for chains of Scrum teams. International Journal of Information and Software Technology, 57,1(1/2015), p.52-65.
Yeo02	Yeo, K.T., Critical failure factors in information system projects. International Journal of Project Management, 20,3(4/2002), p.241-246.

Appendix 1. Categorization of Atomic Challenges of Large Scale Software Engineering Project or IS Project

Problem category	Challenges / problems
1. Sharing the same understanding across large organization	1:1 Agreeing on common definitions used in requirements definition 1:2 Agreeing on common design principles, 1:3 Agreeing on coding standards over multiple teams 1:4 Agreeing on the approval and completion criteria for deliverables when moving to testing 1:9 Agreeing on common modelling languages and methods to needed extent between teams 1:10, 2:10 Agreeing and sharing the same criteria and standards for quality over multiple/distributed teams 3:9 Shared models are required over split features and their boundaries 3:9 Need to recognize what must what is critical to understand and be modelled 4:9, 6:9 Updating and communicating updated shared models after changes
1. Setting roles and responsibilities over multiple teams	1:1 Deciding the best organization structure for requirements definition process 1:4 Distribution of testing responsibilities over teams and to common testing organization 1:7 Defining organizational setting which facilitates engineering processes and coordination over functional areas
3. Distributing and assigning tasks for multiple teams	1:2 Distribution of design tasks to teams 1:3 Dividing the implementation work to teams
4. Decision making over multiple teams	1:12 Making prioritization and scoping decisions and tools/component selections which have different impacts over multiple teams 1:7 Organizing decision making and right participants over multiple teams 3:12 Prioritization of split features in the context of each part
5. Communication over multiple / distributed teams	2:1 Requirements negotiation, communicating requirements with distributed teams 2:2 Communicating design to/from distributed teams 2:4 Communicating requirements, 2:4 Communicating test results/incidents with distributed teams 2:6 Communicating software configuration status to distributed teams 2:7 Communicating the progress of distributed teams 1:7, 2:7 Ensuring knowledge sharing over distributed teams 2:9 Tool support for sharing models and other deliverables with distributed teams
6. Coordination and dependency management over multiple / distributed teams	1:6 Coordinating software configuration with multiple teams Alignment of testing activities with software configuration status and environments 2:7 Coordination of the distributed teams regards common milestones and target schedules 1:7 Monitoring the total progress. 3:1 Splitting large features to smaller sub-features and making architectural decisions impacting widely in the system landscape 3:1 Recognition of dependencies and boundaries regarding split features 3:2 Creating design of interfaces/interactions related to split features. 3:2 Communicating the design regards to split features and architecture decisions, 3:2 Synchronizing the design work of split features 3:3 Synchronizing the implementation work of split features 3:3 Integration and integration testing of split features

	3:4 Following up and coordinating completion of split features for testing, 3:4 Organizing E2E testing of large features split involving experts of multiple teams 3:6 Keeping the dependencies when planning releases and managing builds including large split features. 3:6 Keeping software configuration working in situations involving split features 3:6 Planning timing and meaningful content for releases 3:7 Coordinating schedules and deliverables over split features. 3:7 Monitoring progress and completion of split features and completion of the feature.
7. Dealing with changes and unpredictability	4:1 Long time span increases the amount of changing requirements 4:2, 4:3 During long time span changes may be inflicted to designed or completed features 4:4, 6:4 Keeping requirements up to date during long time span 4:4 Defining verification and approval criteria for requirements changing during the time span 4:6 Planning timing and meaningful content for releases while changes to implemented features may already be known 4:7 Expected changes during long time span lower the credibility of the plans created in the initiation phase 4:7 Changes during long time span cause lots of re-planning. 4:7 Measuring success of project after lots of changes is difficult 4:12 Changing business goals and priorities are possible during long time spans which impact the project feasibility, scope and success 5:1 Information system inherent complexity causes incomplete and changing requirements 5:2 Incomplete and changing requirements cause design changes 5:3 Incomplete and changing requirements cause changes during implementation time 5:4 Keeping requirements up to date while completion during development time 5:4, 6:4 Defining verification and approval criteria for changing requirements 5:4 Since not all conditions can be tested, it is difficult to decide readiness for approval 5:6, 6:6 Due to late finalization of requirements release content may not be fixed until nearly release time 5:7 Due to IS domain complexity final solution can't be fully defined in the initiation phase hence not all coming activities are known in initial planning phase, causing incomplete plans (schedule estimates, resource needs, recognised work packages and tasks, etc). 5:7 Incomplete plans require updating and re-planning. 5:12 IS complexity and inability to model everything adds uncertainty in decision making 6:1 Due to unpredictable nature of problem solving process requirements may stay incomplete and changing and requirements engineering activity can't be completed before late in the development phase. 6:2 Due to unpredictable nature of problem solving, Requirements definition, design and implementation are intertwined and can't be completed before completion of development and approval of the feature 6:3 Due to unpredictable nature of problem solving, development completion time may be difficult to predict before it's completed with verification and approval. Even after approval defects can be found causing changes to the design and implementation 6:4 Time and needed test rounds for feature can't be predicted, scheduling the approvals are difficult.

	6:7 Due to unpredictably nature of software development work (problem solving) all activities needed in the design and implementation phases can't be recognized in the initial planning causing incomplete plans. Incomplete plans require updating and re-planning 6:10 Deciding when and how to measure quality when end results and requirements are not known/fixed until late stage 6:12 Unpredictable nature of software development adds uncertainty in decision making
8. Dealing with large amount of "customers"/stakeholders	7:1 Large amount of requirements and requirements sources/stakeholders that need to be involved and satisfied 7:3, 7:4, 7:10 Validating ad confirming the results with many stakeholders 7:12 Decision making is difficult with various stakeholders having contradicting objectives
9. Interacting with parallel software maintenance (or other organizational processes)	3:5 Agreeing the incident management and maintenance responsibilities over large features involving several subsystems and possibly several maintenance organizations 4:5 Long development project may be still ongoing while maintenance process needs to be set up and the interaction between these two needs to be planned (in regards to functionalities changed in both work streams and timing of changes near releases) 4:5 Creating the documentation for maintenance when lots of content from long development project and changes still coming.
10. Personnel/human resources and sourcing decisions e.g. offshoring/outsourcing and personnel changes	2:12 Need to make decisions over offshoring/outsourcing 4:11 Personnel changes likely during long time span, learning time and group dynamics aspects may have impact when personnel is changing.

Table 8: Categorization of atomic challenges of large scale software engineering project or IS project.

Appendix 2. Mapping of Problem Categories to SWEBOK Knowledge Areas and Large Scale Project Features

SWEBOK Knowledge Area	**Features**						
	Multiple teams	**Distributed teams**	**Large & spanning features**	**Long time span**	**Software / IS complexity**	**Problem solving process nature**	**Other**
Software requirements	1:1 (1)Sharing the same understanding cross multiple teams (2)Setting roles and responsibilities over multiple teams	2:1 (5)Communication over multiple / distributed teams	3:1 (6)Coordination and dependency management over multiple / distributed teams	4:1 (7)Dealing with changes and unpredictability	5:1 (7)Dealing with changes and unpredictability	6:1 (7)Dealing with changes and unpredictability	7:1 (8)Dealing with large amount of "customers"/stakeholders
Software design	1:2 (1) Sharing the same understanding cross multiple teams (3)Distributing and assigning tasks for multiple teams	2:2 (5)Communication over multiple / distributed teams	3:2 (6)Coordination and dependency management over multiple / distributed teams	4:2 (7)Dealing with changes and unpredictability	5:2 (7)Dealing with changes and unpredictability	6:2 (7)Dealing with changes and unpredictability	7:2 -
Software construction	1:3 (1) Sharing the same understanding cross multiple teams (3)Distributing and assigning tasks for multiple teams	2:3 -	3:3 (6)Coordination and dependency management over multiple / distributed teams	4:3 (7)Dealing with changes and unpredictability	5:3 (7)Dealing with changes and unpredictability	6:3 (7)Dealing with changes and unpredictability	7:3 (8)Dealing with large amount of "customers"/stakeholders
Software testing	1:4 (2) Setting roles and responsibilities over	2:4 (5)Communication over multiple / distributed teams	3:4 (6)Coordination and dependency	4:4 (7)Dealing with changes and unpredictability	5:4 (7)Dealing with changes	6:4 (7)Dealing with changes	7:4 (8)Dealing with large amount of

	multiple teams. (1) Sharing the same understanding cross multiple teams.		management over multiple / distributed teams		and unpredictability	and unpredictability	"customers"/stakeholders
Software maintenance	-	-	3:5 (9)Interacting with parallel software maintenance (or other organizational processes)	4:5 (9)Interacting with parallel software maintenance (or other organizational processes)	-	-	-
Software configuration management	1:6 (6)Coordination and dependency management over multiple / distributed teams	2:6 (5)Communication over multiple / distributed teams	3:6 (6)Coordination and dependency management over multiple / distributed teams	4:6 (7)Dealing with changes and unpredictability	5:6 (7)Dealing with changes and unpredictability	6:6 (7)Dealing with changes and unpredictability	-
Software engineering management	1:7 (2) Setting roles and responsibilities over multiple teams. (4)Decision making over multiple teams. (5)Communication over multiple / distributed teams: (6)Coordination and dependency management over	2:7 (5)Communication over multiple / distributed teams (6)Coordination and dependency management over multiple / distributed teams	3:7 (6)Coordination and dependency management over multiple / distributed teams	4:7 (7)Dealing with changes and unpredictability	5:7 (7)Dealing with changes and unpredictability	6:7 (7)Dealing with changes and unpredictability	-

	multiple / distributed teams.						
Software engineering process	1:8 Selection and tailoring of processes and lifecycle models to support features of large scale project						
Software engineering models and methods	1:9 (1) Sharing the same understanding cross multiple teams	2:9 (5)Communication over multiple / distributed teams	3:9 (1) Sharing the same understanding cross multiple teams	4:9 (1) Sharing the same understanding cross multiple teams:		6:9 (1) Sharing the same understanding cross multiple teams:	-
Software quality	1:10 (1) Sharing the same understanding cross multiple teams	2:10 (1) Sharing the same understanding cross multiple teams:	-	-	-	6:10 (7)Dealing with changes and unpredictability	7:10 (8)Dealing with large amount of "customers"/stakeholders
Software engineering professional practice	1:11 -	-	-	4:11 (10)Personnel/human resources and sourcing decisions e.g. offshoring/outsourcing and personnel changes	-	-	-
Software engineering economics	1:12 (4)Decision making over multiple teams	2:12 (10)Personnel/human resources and sourcing decisions e.g. offshoring/outsourcing and personnel changes	3:12 (4)Decision making over multiple teams:	4:12 (7)Dealing with changes and unpredictability	5:12 (7)Dealing with changes and unpredictability	6:12 (7)Dealing with changes and unpredictability	7:12 (8)Dealing with large amount of "customers"/stakeholders
Computing foundations							
Mathematical foundations							

Engineering foundations							

Table 9: Mapping of problem categories to SWEBOK Knowledge Areas and large scale project features.