A STUDY ON APPLICABILITY OF THE SCRUM FRAMEWORK FOR LARGE SOFTWARE PROJECTS

by

Jaweria Sultana

Bachelor of Engineering

Osmania University, India, 2006

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Canada, 2015

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

A STUDY ON APPLICABILITY OF THE SCRUM FRAMEWORK FOR LARGE SOFTWARE PROJECTS

Jaweria Sultana

M. Sc., Computer Science, 2015

Ryerson University, Toronto, Canada

Abstract

The primary objective of this research is to investigate the adaptability of the Scrum framework for large scale projects. A two phase approach has been undertaken towards the goal. The first phase involves conducting a systematic literature review to identify and elaborate scaling practices used in the current industry. The review also identifies the challenges faced by the developers when the Scrum framework is used for the development of large projects. The second phase involves the construction of a simulation model to analyze the dynamic behavior of the Scrum framework for large projects. The systematic literature review revealed that the major challenge while scaling Scrum is ensuring good communication among project members. The communication overhead was incorporated in the system dynamic model of the Scrum framework. The simulation results showed that there is a reduction in work rate when number of personnel is increased due to the increasing communication overhead.

Acknowledgments

I would like to express my sincere gratitude to my supervisor Dr. Vojislav B. Mišić, for his continuous support, patience, motivation and encouragement throughout my graduate studies. His guidance helped me throughout my research and in writing this thesis. It was a great privilege to work under his guidance. I am also thankful to the Department of Computer Science and the School of Graduate Studies at Ryerson University for all the financial and infrastructural assistance. Lastly, I would like to thank my parents, my husband and my little kids for all their moral support and love.

Dedicated to my parents

Contents

Chapter 1

Introduction

1.1. Motivation

Agile methods have proven to be beneficial in small organizations and there has also been growing interest in using these methods in large organizations. Using agile methods to develop large systems presents a challenging set of issues [49]. In order to produce lots of software quickly involving large teams, the agile methods involved must be scaled to meet the task. Scaling agile methodologies requires some changes to original method to accommodate large teams. It results in several challenges that has to be figured out and resolved in order to achieve same quality and productivity these methods gives for smaller projects.

In this thesis, we selected Scrum which is an agile software development framework to work upon. Scrum is the most popular agile approach used for developing software [27]. Scrum is a lightweight process framework for agile development and was basically designed for smaller projects with limited team size [52]. However, in recent years it is widely used for large and distributed projects as well [33] [23] [44] [22].

Scrum emphasizes on collaboration, functioning software, team self-management, and the flexibility to adapt to emerging business needs. It employs concepts such as self-directed co-located teams, time-boxed sprints (duration of work), and regular customer feedback from working software.

Adapting Scrum for large projects involving many members (greater than ideal size which is between 7 to 9) requires scaling the Scrum framework as it will be difficult to collaborate and communicate in large teams. There is a scalable version of Scrum called as Scrum of Scrums (SOS) which can be used for large projects [36]. According to SOS, in situations where the Scrum team size exceeds ten people, multiple scrum teams can be formed to work on the project.

It uses frequent Scrum of Scrums meetings for inter team coordination and collaboration [36]. How these meetings are coordinated and applied is not reported in [36]. No effective techniques have evolved to co-ordinate the work of multiple scrum teams and manage dependencies between them. The SOS approach holding meetings with team's representatives becomes more time consuming and unmanageable as the number of teams multiplies. There are lot of empirical studies which focused on scalability issue of Scrum. But only few of them discussed about resultant quality and productivity obtained through scaling Scrum. Most of these studies are carried out through interviews, questionnaires and case studies involving software developers. In this thesis, we investigate on project management challenges faced by organizations when they use Scrum for large projects. In addition to this we also outline scaling practices that are beneficial for such kind of projects. We also designed several causal loops to investigate relationships between various factors affecting resultant quality and productivity of the Scrum framework. Using these causal loops, System dynamic model was constructed to analyze the behavior of the scaled Scrum framework. These findings might help developers to decide the suitability of the Scrum for their project depending upon project size, members involved and the deadline. It can direct the developers to focus on the right issues from the beginning of the project which in turn can reduce development effort and cost.

1.2.Research Approach

This thesis focuses on the applicability of Scrum in large scale software development organizations. In this context, a large scale means involving many developers more than ideal size (7 to 9 people) for agile development and can be distributed or not. The objectives of this study are to investigate management challenges in adopting scrum for large scale projects and to find and elaborate scaling practices used. To make this study more specific, we try to answer following research questions through this work:

1. Why to scale Scrum when used for large projects?
2. What are the scaling practices currently used by the industry?
3. Which Scaling practice is widely used and why?
4. What changes are required to the original Scrum framework to make it adaptable to large projects and what effect does that have on quality and productivity?
5. What are the challenges faced by developers in adopting Scrum for large projects?

In order to answer above questions, we adopted a two-phase approach. The first phase involves conducting a systematic literature review of past empirical research. We included literature from the past ten years for the review. The main goal of this review is to investigate adaptability of Scrum for large projects by identifying currently used scaling practices and challenges faced while adapting them.

The second phase involves the construction of a system dynamics simulation model of the Scrum framework. First part of this phase involves developing causal loop diagrams illustrating various factors affecting the Scrum framework. There are several factors that affect productivity, development effort, velocity and quality and each are interrelated to one another. Causal loops can help in determining the relationships between various factors. These causal loops and the results obtained from Systematic literature review are used to develop system dynamics model which constitutes the second part.

1.3.Thesis Contribution

The main contributions of this thesis are:

- A Systematic literature review that investigates the scalability issue of the Scrum framework in detail which gives an overview of the current literature at a glance. The aim of this review is to identify various scaling practices used to scale Scrum and to identify challenges in adopting them. The studies included in SLR will be analyzed using thematic analysis.
- The System dynamic simulation model is built to examine the relationships between various factors that affect the project performance in large scale Scrum. The model is developed iteratively so that each module can also be used separately depending upon the area of the interest. The model can help developers in predicting the project completion time and development cost when calibrated with the project specific parameters.

1.4.Thesis Organization

The remainder of this thesis is organized as follows:

Chapter 2 presents a detailed introduction to the Scrum framework. It is followed by a brief discussion on the scalability issue of the Scrum. It also presents an introduction to systematic literature review methodology. It ends with a review on similar kinds of research works.

Chapter 3 details the systematic literature review conducted to investigate scalability issue of the Scrum framework and also presents the results of the review.

Chapter 4 presents the system dynamic model beginning with an overview of system dynamics modeling followed by causal loop diagrams and iterations of the model. The results obtained from the SD model are also analyzed as the concluding part.

Chapter 5 summarizes the thesis and concludes it by providing suggestions for future work.

Chapter 2

Background and Related Work

This chapter describes the background needed to understand the Scalability issue of the Scrum. First Agile software development is briefly discussed, followed by the Scrum framework and Scrum for large projects in detail. Lastly, it reviews the need of a systematic literature review and similar studies in this field.

2.1. Agile Software Development

In 2001, several agile thought-leaders agreed on what they called the Agile Manifesto [6] stated as:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- ***Individuals and interactions over processes and tools***
- ***Working software over comprehensive documentation***
- ***Customer collaboration over contract negotiation***
- ***Responding to change over following a plan***

That is, while there is value in the items on the right, we value the items on the left more.

Some of the principles behind the Agile Manifesto are:

- Customer satisfaction by rapid, continuous delivery of useful software
- Working software is delivered frequently (weeks rather than months)
- Working software is the principal measure of progress
- Even late changes in requirements are welcomed

- Close, daily cooperation between business people and developers
- Face-to-face conversation is the best form of communication (co-location)
- Projects are built around motivated individuals, who should be trusted
- Continuous attention to technical excellence and good design
- Simplicity
- Self-organizing teams
- Regular adaptation to changing circumstances

These principles govern all the techniques and rules in the different agile methods, which all strive to make software development more flexible and overall more successful. The manifesto bred a movement in the software industry known as agile software development. Agile software development refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams. The various agile methodologies share much of the same principles, as well as many of the same characteristics and practices. But from an implementation viewpoint, each has its own set of practices, terminology, and strategies. Well-known agile software development methods and/or process frameworks include:

- Extreme programming (XP)
- Scrum
- Feature-driven development (FDD)
- Lean software development
- Kanban software development
- Adaptive software development (ASD)
- Agile modeling
- Agile Unified Process (AUP)

- Crystal Clear Methods
- Dynamic systems development method (DSDM)

In 2009 and 2010 Forrester Inc. surveyed 1298 and 1093 software professionals respectively, requesting them to select the methodology that most closely reflected their development process [64].The respondents identified some of the most in-use agile methodologies, with Scrum being the most popular. In the following section Scrum framework is discussed in detail while other agile methodologies are not discussed being out of the scope of this thesis.

2.2. The Scrum Framework

Scrum was introduced by Takeuchi, DeGrace, Schwaber, and others in the late 1990s [52]. Scrum is an agile management framework for incremental product development using one or more cross-functional, self-organizing teams of about seven people each. It is an iterative, time-boxed, incremental project management method based on a simple "inspect and adapt" framework [25]. The Scrum framework is based on a set of values, principles, and practices that provide the foundation for the project management. It emphasizes decision making from real-world results rather than assumptions. It provides a structure of roles, meetings, rules, and artifacts. Teams are responsible for creating and adapting their processes within this framework.

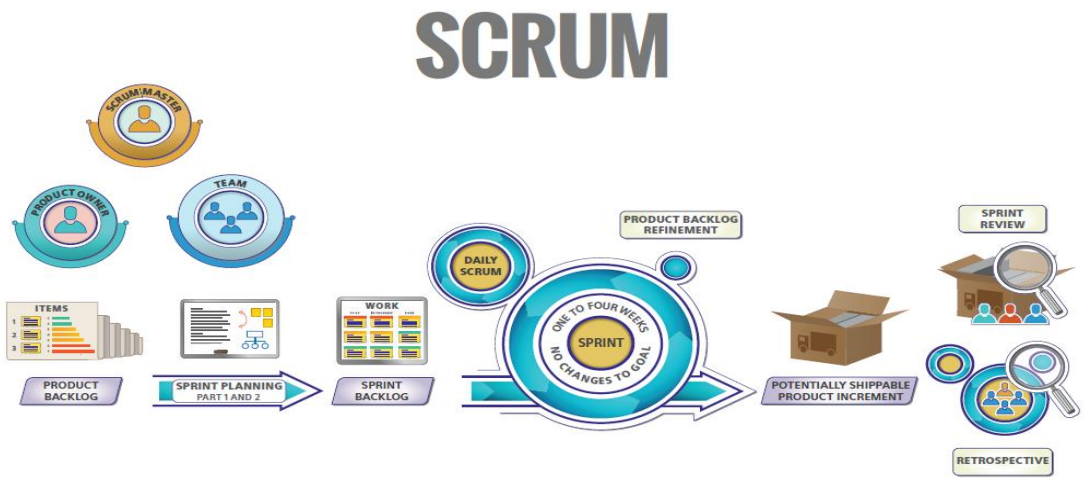

Figure 1 Overview of Scrum (Adapted from the Scrum Premier 2.2[16])

Scrum uses fixed-length iterations, called Sprints, which are typically two weeks or 30 days long. The Sprints are *time boxed* – they end on a specific date whether the work has been completed or not, and are never extended [16]. Usually Scrum Teams choose one Sprint length and use it for all their Sprints until they improve and can use a shorter cycle. Scrum teams attempt to build a potentially shippable product increment on iteration. The Scrum framework, shown in Figure 1, is first initiated by collecting requirements in the form of user stories from the customers, the teams and other stakeholders. All of these user stories are arranged in priority order in the Product backlog. The high priority items are selected from the product backlog to be implemented in the next Sprint. During the Sprint, no new items may be added. At the start of the Sprint, a planning meeting is held where team members figure out how many items they can commit to, and then create a sprint backlog – a list of the tasks to perform during the Sprint [16].

During each Sprint, the Scrum team works on development of the requirement including its design, coding, testing to yield fully implemented functionality. At the end, these features are coded, tested and integrated into the evolving product or system.

For every Sprint, a daily scrum meeting is held for discussing the current progress and impediments. The result of the each Sprint is a potentially shippable product increment. Thus, each Sprint provides a working functionality for the product. At the end of the Sprint, the Team reviews the Sprint with stakeholders, and demonstrates what it has built. Feedback obtained from participants is incorporated in the next Sprint [16] [32] [51].

The significant benefits Scrum can deliver to business are reflected by the companies that had adopted it which includes Microsoft [7], Yahoo! [12], Nokia [2], Intel [22] and the Ericsson [23].

According to Schwaber and Beedle [52] there is a list of practices to follow in order to use Scrum. Schwaber and Beedle [52] divide these practices into seven categories: the Scrum Master, Product Backlog, Scrum Teams, Daily Scrum Meetings, Sprint Planning Meeting, Sprint, and Sprint Review.

The subsequent sub-sections reveal the principals and practices behind the power of Scrum.

2.2.1. Product backlog

The Product Backlog is a prioritized list of project requirements with estimated times to develop them into completed product functionality. The Product Backlog is continuously updated to reflect changes in the needs of the customer, new ideas or insights, moves by the competition, technical hurdles that appear, and so forth [16]. The product owner is responsible for prioritization of the product backlog items after consulting from the team and stakeholders involved. All the entries within the Product Backlog have to be estimated in terms of either story points, function points or simply point. This estimation can then be used to prioritize entries in the Product Backlog and to plan releases. The Team and the Product Owner decides the effort estimate and technical risk estimates for each item in the product backlog.

The larger items in the product backlog are broken into smaller items and assigned individual priorities. This is referred to as product backlog refinement. The Product Backlog items for the future Sprints should be small and fine-grained enough that they are well understood by the team and they make near to concise estimates [16].

Figure 2 shows a sample product backlog template.

ToDo List

ID	Story	Estimation	Priority
7	As an unauthorized User I want to create a new account	3	1
1	As an unauthorized User I want to login	1	2
10	As an authorized User I want to logout	1	3
9	Create script to purge database	1	4
2	As an authorized User I want to see the list of items so that I can select one	2	5
4	As an authorized User I want to add a new item so that it appears in the list	5	6
3	As an authorized User I want to delete the selected item	2	7
5	As an authorized User I want to edit the selected item	5	8
6	As an authorized User I want to set a reminder for a selected item so that I am reminded when item is due	8	9
8	As an administrator I want to see the list of accounts on login	2	10
Total		**30**	

Figure 2 Example of Product Backlog [28]

2.2.2. Sprint backlog

The Sprint backlog is a list of tasks identified by the team to be completed during the upcoming Sprint. During Sprint planning meeting, the team selects the highest priority items from the product backlog, usually in the form of user stories, and identifies the tasks necessary to complete each user story. Most teams also estimate how many hours each task will take someone on the team to complete [52] [16]. Only the team is authorized to change the item selection within the sprint [52]. The tasks should be detailed in terms of man-hours. The tasks are measured in hours whereas the product backlog items are measured in relative story points. The tasks include information about the work that has to be accomplished. Sometimes Sprint backlog is linked with the product backlog to trace the progress of the product backlog items [52].

The Sprint Backlog can be kept electronically within e.g. an Excel-Sheet or with cards on a task board. An example of Sprint backlog in a spreadsheet is shown in Figure 3.

User Story	Tasks	Day 1	Day 2	Day 3	Day 4	Day 5	...
As a member, I can read profiles of other members so that I can find someone to date.	Code the ...	8	4	8	0		
	Design the ...	16	12	10	4		
	Meet with Mary about ...	8	16	16	11		
	Design the UI	12	6	0	0		
	Automate tests ...	4	4	1	0		
	Code the other ...	8	8	8	8		
As a member, I can update my billing information.	Update security tests	6	6	4	0		
	Design a solution to ...	12	6	0	0		
	Write test plan	8	8	4	0		
	Automate tests ...	12	12	10	6		
	Code the ...	8	8	8	4		

Figure 3 Example of Sprint backlog [65]

During the Sprint, team members are expected to update the Sprint backlog as the tasks are fulfilled. Many teams will do this during the daily scrum. Every day, the estimated work remaining in the sprint is calculated and graphed by the Scrum Master in Sprint Burnout chart illustrated in following section.

2.2.3. Sprint Burn-down Chart

Sprint Burn-down Chart is a graph used by the teams to track the development effort remaining in a Sprint. It shows, each day, a new estimate of remaining work until the Team finishes it. It is called a burn-down chart as it is downward sloping graph and reaches zero by the last day of the Sprint [16]. Every day the team member's work on tasks and the work should decrease every day. Usually on horizontal axis, time or number of Sprints are plotted while on vertical axis, work remaining (Story points or man-hours) is plotted.

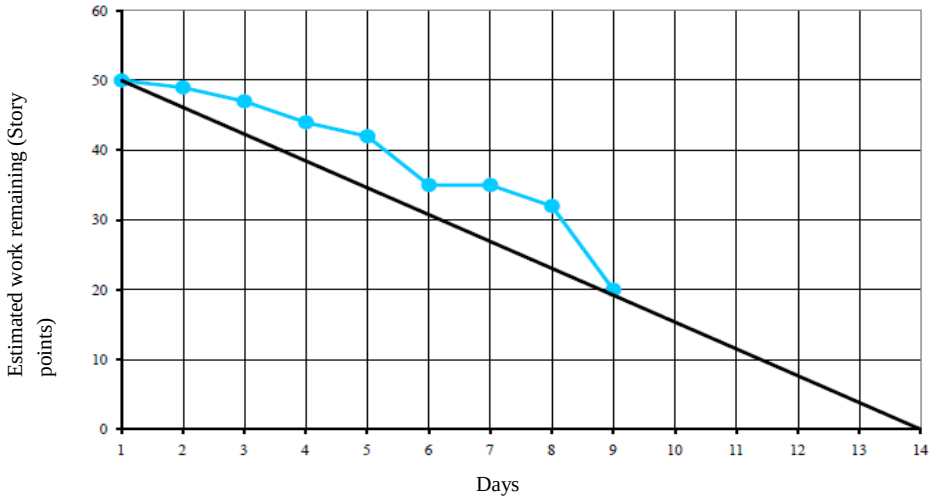

Figure 4 Sprint Burn-down Chart [16] [58]

The above chart in Figure 4 shows a Sprint burn-down chart for two weeks (14 days). On X-axis days are plotted while on Y-axis estimated work remaining is plotted in terms of story points. The unit used for vertical axis is decided by the team. Every day this chart is updated and dots are joined to form a line. This line is called as burn-down line. The linear line shown is the idealized line. It represents a linear progress from day one to the end of the sprint. In the graph shown above, 50 story points have been selected from the product backlog for the current sprint. According to the linear line, the remaining work is reached zero at the last day of the Sprint [16]. In Figure 4, since the burn-down line is above the idealized line, it can be inferred that the team was first behind the estimation. However, the idealized line is reached at the day nine, and the work remaining is 20 story points.

There is another chart used in the Scrum framework called as release burn-down chart which shows the remaining work in the release. The release burn-down focuses on the requirements rather than tasks. On the horizontal axis Sprints is plotted instead of days [16]. It means that Sprint burn-down shows the status of Sprint backlog tasks while release burn-down shows the status of the product backlog items.

2.2.4. Sprint Planning Meeting

At the beginning of each Sprint, the Sprint Planning Meeting takes place. The maximum duration of the meeting is five percentage of the Sprint length. The inputs to the Sprint planning meeting are the team capacity and the product backlog [50]. The meeting is divided into parts, each also time-boxed. During the first part, the Product Owner presents the highest priority Product Backlog items to the team. The team and the Product Owner collaborate to help the team estimate how much Product Backlog it can develop into working software in the upcoming Sprint. The second part of the meeting focuses on how to implement the items that the team decides to take on. The team estimates the number of items from the product backlog they can complete by the end of the Sprint. The higher priority items are first selected for the implementation.

2.2.5. Daily Scrum

A daily Scrum meeting is a short 15-minute meeting. In the Daily Scrum each member of the Team reports three things to the other members of the Team [16]:

1. What has been accomplished since the last meeting?
2. What will be done before the next meeting? and
3. What obstacles are in the way?

The Daily Scrum improves communications, collaboration and knowledge about the current progress. The Scrum Master is responsible for effectively organizing the Daily Scrum.

2.2.6. Sprint Review and Retrospective

After the Sprint ends there is a meeting where people review the Sprint. The Product owner, ScrumMaster, team and all the stakeholders participate in this meeting. In this review what has been accomplished in the Sprint is discussed. The Sprint Review is an inspect and adapt activity for the product [16]. The Sprint review should be no longer than 30 minutes [16].

The Sprint Retrospective, which follows the Review, is an inspect and adapt activity regarding the process and environment. In this meeting, team discusses which practices are proving beneficial and which are not. It is time boxed for 45 minutes per week of Sprint [16].

2.2.7. Scrum Role

The Scrum framework includes three roles, Product Owner, Team, and Scrum Master.
The Product Owner is the key stakeholder and is responsible for return on investment. The Product owner prioritizes the product backlog during the Sprint planning meeting. She/He is responsible for the product backlog being updated, visible, and prioritized all the time [16].

The Team in Scrum is cross functional; for a software product the Team might include people with skills in analysis, development, testing, interface design, database design, architecture, documentation, and so on [16]. It includes all the expertise necessary to deliver the potentially shippable product each Sprint and it should be self-managing [16]. The Team decides how many items from product backlog to be selected to implement in a Sprint. The Team in Scrum is seven plus or minus two people [52] [16].The Team works on requirements to develop functional product and also provides ideas to the product owner to enhance productivity.

The Scrum Master helps the team learn and apply Scrum to attain business value [16]. As the team is already a self-organizing team, the Scrum Master enhances self-management, cross functionality, creativity and empowerment [18] [51]. The Scrum Master is the facilitator of the meetings and is responsible for ensuring that the team members are able to proceed in their tasks.

2.3. Scaling Scrum for Large projects

The Scrum Framework described in [6] [52] [16] works best for a single co-located Scrum team. However, nowadays the projects and resources are growing enormously and small size team cannot accomplish the goals in limited time. As a consequence the number of members involved has to be increased and/or the teams can be distributed. The reasons for this can be unavailability of experts, project size being too big, low cost usage of resources in different countries or speed up work by utilizing different time-zones.

According to Abrahamsson [2], Scrum is an effective approach for project management with small, co-located development teams. However, Sutherland and Schwaber [61] argue that Scrum can also be used for large and distributed teams.

In this thesis, Scaling refers to tailoring original Scrum framework to make it adaptable for large projects. Larger projects are those which involve more number of people (more than the ideal size) and may be distributed or not.

According to Schwaber [52] following are the reasons for scaling Scrum:

1. Planning to fulfill functionality more quickly by applying more number of Scrum Teams to the Product Backlog.

2. When more people are needed for one or more Sprints of a Product backlog than required for a single Scrum team. A singularity of diverse skills applied at one time may generate this need, such as developing a user interface framework, secure architecture, and piece of functionality within one Sprint [52].

All of the basic principles, artifacts, values, roles, and meetings of Scrum have to be considered, whether Scrum is singular or scaled.

2.3.1. Options for Scaling Scrum

In this section, we describe different ways in which Scrum can be scaled to be used for large projects.

A typical Scrum team consists of 6-10 people but Jeff Sutherland has successfully scaled Scrum up to over 800 people [60]. The primary way of scaling Scrum to work with large teams is to coordinate a Scrum-of-Scrums or a so called Meta-Scrum. With this approach each Scrum team proceeds as normal but to coordinate the work of multiple teams , Scrum of Scrums meetings are held in which only team's representative participates. These meetings are analogous to the Daily Scrum meeting but do tend to happen weekly rather than daily.These meetings allow teams to discuss their work, focusing especially on areas of overlap and integration. In many organizations, having a Scrum of Scrums meeting two or three times a week is sufficient.

The problems arise when there is one large product backlog to be divided among teams. It would be difficult to manage by a single product owner. If multiple product backlogs are used, then the concept of the area product backlogs can be utilized. The interconnected teams are grouped to form areas. The product owner's responsibility is now distributed among multiple people, i.e., to the area product owners (APO). Each requirement area includes a backlog, i.e., the area product backlog. The areas are related to features and business, not to the product architecture [35]. Figure 5 shows the scaled Scrum which uses area product owner. This framework requires very good coordination among teams in order to complete the project successfully.

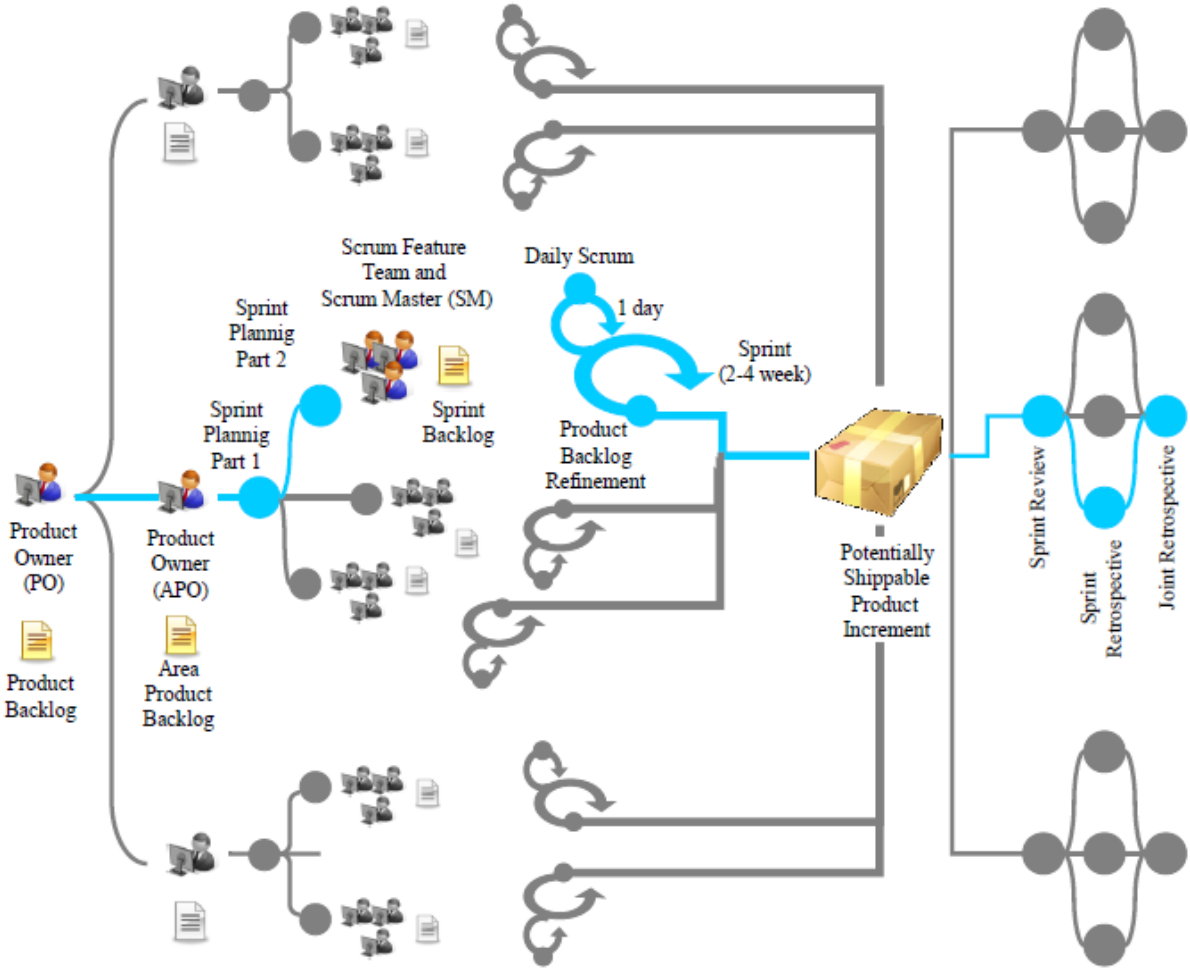

Figure 5 The Scaled Scrum framework (Adapted from [35])

2.4. Systematic Literature Review

A Systematic literature review (SLR) is a secondary study for identifying, evaluating and interpreting all available research relevant to a particular research question or topic of interest [31]. Individual studies involved in a systematic review are called *primary* studies whereas a systematic review is a form of a *secondary* study. Case studies are commonly used in most of the software engineering research, and systematic review can discover and synthesise new results by grouping several such similar studies.

The most common reasons for conducting Systematic literature review according to [31] are:

- To summarize the existing evidence concerning a treatment or technology e.g. to summarize the empirical evidence of the benefits and limitations of a specific agile method;
- To identify any gaps in current research in order to suggest areas for the further investigation; and
- To provide a framework or background for appropriately positioning of new research activities

However, systematic reviews can also be undertaken to examine the extent to which empirical evidence supports/contradicts theoretical hypotheses, or even to assist the generation of new hypotheses.

2.5. Related Work

This section presents the summary of similar research carried out in this field using Systematic literature review and System dynamics modeling.

Hussain et al [27] conducted a systematic literature review of the primary studies that reported using Scrum practices in global software development (GSD) projects. The extracted data from these studies were used to identify various challenges of using Scrum in GSD. Its only focus was to study globally distributed projects. They concluded that the use of Scrum practices in GSD was limited by project's contextual factors. The review findings also reveal that the temporal, geographical and socio-cultural distance in distributed projects creates a number of challenges

towards GSD communication, coordination and collaboration processes. The most important among them is communication challenge. In order to use Scrum for GSD, it have to be modified or extended in a way to overcome these challenges.

The Systematic literature review of agile methods carried out by Dybå and Dingsøyr [17] in 2008 identified 33 primary studies. Their focus was on empirical studies which adopted agile development. They categorized the studies into four categories namely introduction and adoption, perceptions of agile methods, human and social factors, and comparative studies. Extreme programming was found to be the most prominent agile method in terms of citations and also adoption. The Scrum framework was found to be the most useful method for project management.

Cardozo et al [11] carried out systematic literature review to find the scientific evidence of the correlation between the use of Scrum and productivity in Software Projects. It was not clear from this paper that whether the type of projects considered was small or large. However they reported considerable improvement in productivity when Scrum was adopted. This SLR reported a failure in finding sufficient evidence to consider other project aspects, such as communication, reliability, cohesion, and business value, as a significant outcome, even though they seem to be incidental benefits, of using Scrum.

In [21], Glaiel et al presented an Agile Project Dynamics model that captured the agile genes as a separate component of the model and allows experimentation with combinations of practices and management policies. The agile genes were identified as: Story/feature driven, iterative/incremental, refactoring, micro-optimizing, customer involvement, team dynamics, and continuous integration. The goal of this model was to gain insights and recommendations to integrate agile practices into a large-scale software engineering organization. Glaiel et al concluded that team Dynamics, feature-driven, and iterative-incremental genes are relatively easy to implement or adopt, as most of these practices dictate the behavior of the software development team alone, and do not require much buy-in from other stakeholders in the product development organization. Whereas the advanced genes such as Continuous Integration and Customer Involvement require much more coordination and also require buy-in from stakeholders.

Cocco et al in [13] developed a simple system dynamics model for describing the behavior of Waterfall, Scrum and Lean-Kanban Software development methods. The traditional Waterfall model was compared by means of simulation techniques with two agile and less prescriptive process tools, Scrum and Lean-Kanban. The objectives of the study were to identify relationships and mechanisms within a software project in case of three software development processes. They concluded that Scrum and Lean-Kanban performed better than waterfall method. They claimed the resulting behavior of the simulation model to be quite realistic with respect to real projects.

Chapter 3

Systematic Literature Review

This part of thesis presents a systematic literature review on adoption of the Scrum for large scale projects. The Systematic literature review conducted in this thesis has been carried out in three main stages adopted from the guidelines provided in [31] about conducting a systematic review. The three stages are:

1. Planning the review
2. Conducting the review
3. Reporting the review

The Systematic literature review (SLR) begins by identifying the need for a review followed by the development of the review protocol. Each stage and activities associated with it are illustrated in the following sections.

3.1. Planning the Review

In this stage, the main focus is to identify the need to carry out a review. The steps involved are described below.

3.1.1. Identification of need for a systematic review

This review has been conducted to find out applicability of Scrum for large projects. Nowadays Scrum is widely adopted agile framework by software organizations. As stated in introduction part, Scrum was originally developed for small teams involving 7 to 9 members. However since last decade it has been used for development of large projects involving hundreds of team members. It has also been tremendously used in distributed projects where teams are distributed in two or more different countries. There are several challenges faced while adopting Scrum for such large projects which are described in these papers [42] [46] [44]. The main challenge is inter team collaboration and coordination. Most of the available Scrum literature reports the use of scrum of scrums meetings, use of retrospectives, area product owners and sprint demos. Paasivara

et al [42] [46] [44] have illustrated this concept of Scrum adaptability for large and distributed project with lots of case studies. Their research emphasizes on inter-team coordination techniques and scaling practices for Scrum. There are ample of empirical studies carried out to reflect upon scalability issue of the Scrum for large projects. But only few of them have discussed about the resultant quality and productivity. Through this literature review we try to find out which practices in particular are used for scaling Scrum, to what extent these practices are successful and analyze their effect on productivity and quality. We also identify and shortlist challenges faced by developers while adopting Scrum for large projects. The findings could be used to ease up the Scrum adoption by the developers and to focus on the right issues from the beginning of the project.

3.1.2. Development of a Review Protocol

Review protocol is a complete plan for conducting a systematic review and provides a method for primary studies selection [31]. This section defines a review protocol which will be used to carry out the actual study. The protocol is established based on the review process described in the guidelines for performing the systematic literature review [31]. The review protocol which is adapted for this thesis is illustrated below in Figure 6.

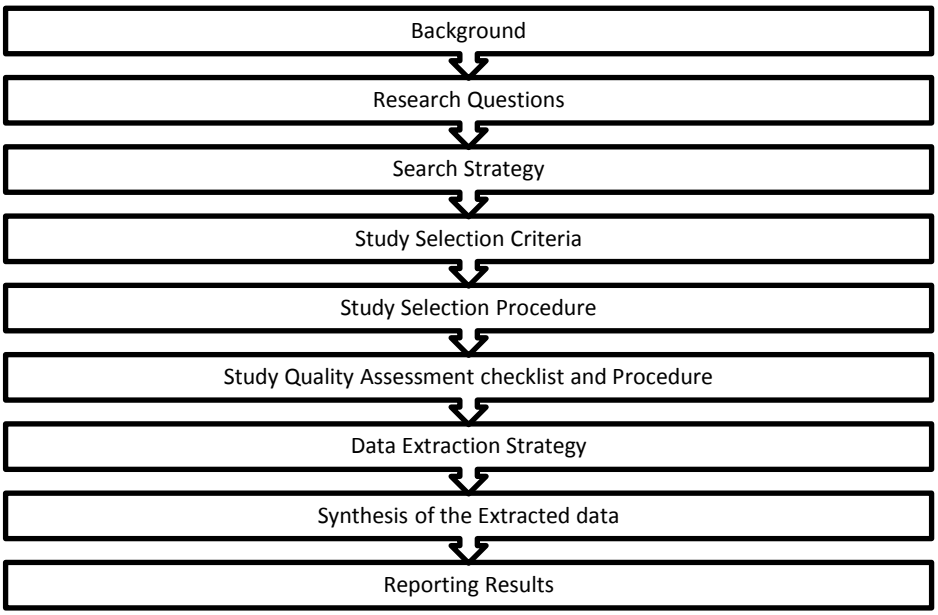

Figure 6. Review Protocol

3.1.3. Background

The main goal of this review is to investigate adaptability of the Scrum for large projects by identifying currently used scaling practices and challenges faced while adopting them. The motivation behind this review is presented in detail in section 1.1 of this thesis. There are several practices adapted by organizations through experience to scale Scrum for large projects. These practices have some effect on quality and productivity of the software developed. The main goal of this review is to gain deeper understanding of such challenges, scaling practices and their effect on software productivity and quality. Through this SLR, we try to find answers to the research questions stated in the following section.

3.1.4. Research Questions

1. Why scale Scrum when used for large projects?
2. What are the scaling practices currently used by the industry?
3. Which Scaling practice is widely used and why?
4. What changes are required to the original Scrum framework to make it adaptable to large projects and what effect does that it has on quality and productivity?
5. What are the challenges faced by developers in adopting Scrum for large projects?

3.2. Conducting the Review

3.2.1. Data Sources and search Strategy

The aim of a systematic review is to find as many primary studies related to the research questions in an unbiased manner. Initial searches aimed at both identifying existing systematic reviews and assessing the volume of potentially relevant studies. The databases that were searched include ACM Digital Library, IEEEXplore, SpringerLink, and Scopus. Each database was queried using the strings 'scaling Scrum,' 'Scrum for large scale projects 'and 'Scaling Agile methodologies' with the search parameters set to look up in the Article Title, Abstract and Keyword fields. We searched for literature published between the years 2004 to 2014. It includes articles from conference proceedings and journals/transactions. The matches from the first stage were reviewed for relevance, which was primarily done by reading through the titles and abstracts and for few papers even the introduction. All research studies that were found to be relevant and those whose relevance was still uncertain were selected for a more detailed analysis in the next stage. In the final stage, all of the selected studies were read and filtered based on the inclusion and exclusion criteria mentioned in the next section. Table 1 lists the number of articles found in first and second stage of screening. The types of papers found were from industry reports, theoretical, empirical and experimental academic papers.

3.2.2. Study Selection Criteria

A study selection criterion is intended to identify those primary studies that provide direct evidence about the research question [31]. Final stage of selection process is based upon the inclusion/exclusion criteria. The Inclusion/ Exclusion criteria are presented below:

Inclusion Criteria

1. Peer reviewed to ensure quality of primary study.
2. Available online to ensure paper accessibility.
3. Research study should be related to at least one research question posed for the review.
4. Paper focuses on adoption of Scrum for large projects either in collocated or distributed environment.
5. The paper has enough empirical data about the project such as number of teams and members allocated for each team.
6. The paper should address scaling practices used only for the Scrum rather than some other agile methodology.

Exclusion Criteria

1. Duplicate copy of the same research study or multiple publications of the same study.
2. Papers which lack numerical data about team members.
3. Any paper that does not possess any of the inclusion criteria was excluded.

Table 1 presents the selected papers in different stages of selection process. Final selection was based on inclusion/exclusion criteria. After a rigorous search and reading of titles and abstracts, we found a total of 798 papers. These papers were further read with introduction and few more sections when required to extract 34 papers. Only studies on projects with more than 10 members were selected for the final review. Finally, 11 papers were selected for a more detailed review using inclusion/exclusion criteria.

Table 1 List of found and selected papers

Database	No. of publications found	1st stage selection	2nd stage selection(Included)
ACM Digital Library	73	10	4
IEEE Xplore	36	9	4
SpringerLink	670	11	2
Scopus	19	4	1
Total	798	34	11

3.2.3. Study quality assessment

In addition to inclusion/exclusion criteria, Study quality assessment is conducted to evaluate the validity of the selected studies. Kitchenham [31] discusses quality assessment with regards to defining the exclusion criteria for the systematic review. Once primary studies are selected, further detailed quality assessment is desired to allow investigators to evaluate differences in the implementation of studies. For detailed quality assessment, checklists can be designed using factors that could bias study results. We prepared a quality assessment checklist shown in Table 2 to assess the quality of selected studies.

Table 2 Quality Assessment Checklist for Selected Studies

No.	Question	Answer
1	Does the research paper illustrate/describe/state that Scrum Methodology is used?	Yes/No
2	Does the study involve large number of personnel and teams?	Yes/No
3	Does the paper explicitly address scaling practices used?	Yes/No
4	Are the data collection methods adequately defined?	Yes/No
5	Are the case studies included in the research paper well addressed empirically?	Yes/No
6	Do the results help in answering the research questions?	Yes/No
7	Are the challenges faced while scaling Scrum discussed?	Yes/No
8	Does the article list any assumptions made?	Yes/No
9	Are any negative results reported?	Yes/No

Most of the selected studies fulfilled answered "Yes" to all the questions listed in quality assessment checklist. The studies which answered less than three "No" were also selected. These studies were included considering their direct relation to the research questions addressed in this SLR.

3.2.4. Data Extraction

During the data extraction phase, the data extraction form was used which was developed in the review protocol. The purpose of the data extraction phase is to extract the relevant data, later to be used to prepare summary tables and to answer research questions. During this stage, data was extracted from each of the 11 primary studies included in this systematic review according to a predefined extraction form. Some of the studies had two or more case studies which are taken as individual studies to form a total of 16 studies.

This form enabled us to record full details of the articles under review and to be specific about how each of them answered our research questions. The Data extraction form is envisioned to integrate current literature-based results and views about Scrum practices used for large scale development. Table 3 shows the Data extraction form which we used for our review to gather data. The columns in this form are populated after reading each paper fully and also reading references wherever required. Our goal was to tabulate as much information as we can in order to extract relevant data for our studies.

Table 3 Data Extraction Form

Study Name	Data Collection Method	Location	No. of Members	No of Teams	Scaling Practices Considered	Negative Feedback	Practices which prove to be Positive	Effect on Productivity/ quality as reported
S1	Interviews (19)	Distributed(4 sites)	170	20	Area Product Owners, Common Sprint Planning, Scrum-of-Scrums, Common Sprint Demo, and Common Retrospective	Common Retrospective not working, SOS and common meetings were not much favored by the participants	APO is helpful whereas Scrum of Scrums were not.	Adoption of scrum resulted in quick releases for customers to test.
S2 (A)	Interviews (15)	Distributed(2 sites)	10	4	Sprint Planning, SOS, Sprint review, and Common Retrospective	Common Retrospective was rarely used.	Sprint Planning and review was used. SOS was modified (weekly status meeting between onshore project manager and offshore team lead)	Not Reported

Study Name	Data Collection Method	Location	No. of Members	No of Teams	Scaling Practices Considered	Negative Feedback	Practices which prove to be Positive	Effect on Productivity/ quality as reported
S2 (B)	Interviews	Distributed Sites (2)	11	2	Sprint Planning, SOS, Sprint Review, Common Retrospective	SOS not used. Common Retrospectives not used. (other meetings and communication opportunities were considered to be sufficient)	Sprint Planning was modified , Sprint Review was used.	Not Reported
S2(C)	Interviews	Distributed Sites(2)	15	5	Sprint Planning, SOS, Sprint review, and Common Retrospective	*Retrospectives* were also tailored. Onshore and offshore Teams conducted retrospectives separately.	SOS, Sprint Planning were modified and used successfully.	Not Reported
S2(D)	Interviews	Distributed(4)	15	4	Sprint Planning, SOS, Sprint review, and Common Retrospective		SOS, Sprint Planning and Retrospectives were modified and used successfully.	Not Reported
S3	Web-based Questionnaire and Survey	Co-located	120	14	Sprint planning, Sprint Demo, Internal Software Documentation, Retrospectives, Open space office, face to face communication	Demos and frequent meeting seem to be stressful for some individuals.	face to face communication, use of task boards increase co-ordination effectiveness.	Increased productivity as well as quality
S4(A)	Interviews	Distributed(4)	160	20	Scrum-of-Scrums	SOS meetings seem to be poor for a team with disjoint interest.	Not reported anything working well in this project.	Not Reported

Study Name	Data Collection Method	Location	No. of Members	No of Teams	Scaling Practices Considered	Negative Feedback	Practices which prove to be Positive	Effect on Productivity/ quality as reported
S4(B)	Interviews (58)	Distributed(2)		25	Scrum-of-Scrums	SOS meetings seem to be poor for a team with disjoint interest. But was successful when tailored with feature specific working teams.	SOS meetings are helpful with teams working on similar feature.	Not Reported
S5	Case Study, Observation	Co-located	49	7	Scrum of Scrums, Feature backlog, Scrum retrospective	More than 3 teams /release were hard to manage	Team focused on values, Commitment, Transparency, Teamwork, No team competition, Knowledge sharing. *SOS meetings helpful	Positive effect on team velocity. Productivity increased.
S6	Interviews	5 teams Co-located, 6 Distributed	88	11	Product backlog	Large number o developers, time pressure, Complex business process rules, Fixed price/final scope dilemma	Maintain a designated backlog of overflow task and technical improvement, Increased testing at Sprint checkpoints, Allocate enough time to describe and Communicate vision.	Factors which increases productivity: High Skills, Anchored methodology, Business and use involvement and Collocation and good infrastructure.
S7(A)	Interviews	Distributed 4	150	20	Scaling Product owner role through Area product owners	Lack of face-face to communication.	APOS should be close to the development team, Sprint planning meetings	Not Reported

Study Name	Data Collection Method	Location	No. of Members	No of Teams	Scaling Practices Considered	Negative Feedback	Practices which prove to be Positive	Effect on Productivity/ quality as reported
S7(B)	Interviews	Distributed 2	170	25	Using proxy product owner as well as product owner. distributed product ownership team (PO team) and all scrum basic activities.	The development teams were not experienced in design and architecture planning,	The PO team had two videoconference meetings each week Locate PO team close to the development teams Team demos to PPOs	Not Reported
S8	Case Study	Collocated	63	9	SOS, Sprint demo, Retrospectives/ reviews	Master product backlog ineffective Agile estimation techniques unsuccessful in estimating points, team velocity. Difficult to make common goal of all team members.	Same sized Sprints, Teams using same tools, Product backlog and product owner for each team. Retrospectives and reviews gave good result.	Not Reported
S9	Case Study/observation	Distributed(3)	210	30	Scrum of Scrums (4 questions), Use of virtual architecture team and more automation.	Challenges in Dependency management, Cross team co-ordination, Scrum of Scrums (4 questions) creating redundancy.	Short and overlapping release cycles,) Provide forums to stimulate collaboration and knowledge sharing between teams. Promote self-organization and decentralized dependency management, virtual architecture team and more automation.	Not Reported

Study Name	Data Collection Method	Location	No. of Members	No of Teams	Scaling Practices Considered	Negative Feedback	Practices which prove to be Positive	Effect on Productivity/ quality as reported
S10	Interviews	Distributed (2)	40	10	Scrum of Scrums, Demos, Separate backlogs for each team.	Video-conferencing difficult for meetings, Misunderstandin requirements.	Synchronized 4-week sprints, weekly SOS, Frequent visits, Unofficial distributed meetings, demos, retrospectives.	Better and improved quality, Better and frequent communication and Improved motivation.
S11	Interviews	Distributed	25 to 100	10	Scrum basics used.	Unclear requirements, limited knowledge of domain, technology, and the organization, and communication problems led to uncertain estimates, unstable plans and integration, and quality problems with the consequent need for rework	Collocation of some members of distributed teams with scrum masters and product owners and regular, well prepared global scrum team meetings improves shared understanding and team coordination, and reduces integration problems	No significant change in quality reported by adopting agile scrum method.

3.2.5. Data Synthesis

Data synthesis involves collecting and summarising the results of the included primary studies [31]. In the data synthesis phase the results from all the findings were tabulated and summarized and each question was assessed individually against the findings. We used data extraction table developed in the previous phase to conduct in-depth analysis. The data was synthesized using thematic analysis, an iterative thematic synthesis process recommended by [15].

The aim of the thematic analysis was to find answers to the proposed research questions. The results obtained from this analysis are detailed in the following results section.

3.3. Results and Analysis

In the first stage of selection, 34 papers were identified which addresses the issue of scalability of Scrum. Each paper was studied in detail by analyzing the context of the study, research questions, and empirical confirmation of the result. One of our main goals was to find studies which reported effect of scaling Scrum on productivity and quality. But we found very few studies reporting the productivity and quality. After thoroughly reading 34 papers, 11 papers were selected based on inclusion/exclusion criteria described in section 3.2.2.

3.3.1. Overview of the selected publications

Only the papers published between the years 2004 to 2014 were selected for the review so that latest information can be gathered. Table 4 shows the classification of the different studies according to the publication year. The gradual increase in number of publications shows the growing interest of practitioners and researchers in this field. However this table shows data only for 11 studies selected for SLR whereas we found more than 2000 studies dealing with this issue in the last ten years.

Table 4 Studies by year of publication

Publication year	2008	2010	2011	2012	2013
Number of selected papers	3	1	3	2	2

The studies S2, S4 and S7 reported results from more than one case study which we take as individual study for our review. Therefore we reviewed results from 16 studies.

3.3.2. Classification based on team's location

The Scrum teams for a large scale project can be co-located or distributed in different places (Cities, States or even Countries). As originally thought that the co-located teams are better to ensure proper communication and deliver more efficiently than distributed teams. But nowadays where global software development is increasing and favourable, teams often consist of people from different parts of the world working together as distributed teams. In our review also, we found most of the studies dealing with scalability issues have case studies which involved distributed teams. Table 5 shows the classification on 16 studies based on team's location.

Table 5 Classification of Studies based on Team's Location

Team's Location	No. of Studies
Distributed	12
Co-located	4

3.3.3. Type of Study

As scalability issue for Scrum deals with the large number of teams, almost all studies found were conducted on industrial scale. We didn't find any of the academic studies dealing with Scrum scalability. All the studies included in this review are conducted in different software organizations.

3.3.4. Research method

The type of studies varied from industrial online survey, case studies, observations, interviews and questionnaires. Table 6 shows the type of study reviewed. 12 studies were classified as using interviews which is the highest among all. Total number of case studies with observation was 3, while only one study used web based questionnaire and survey.

Table 6 Classification of studies based on Research method adopted

Research method	No. of Studies	Studies
Interviews	12	S1,S2(A,B,C,D),S4(A,B),S6,S7(A,B),S10,S11
Case studies/Observations	3	S5,S8,S9
Web-based questionnaire and Survey	1	S3

3.3.5. Number of Project members

Only projects with more than 10 members were selected for review as already stated in section 3.2.2. All the case studies selected for the review had more than 50 members involved except 4 case studies described in paper S2 [28].This paper reports a multi-case study that investigates the impact of key project contextual factors on the use of Scrum practices in GSD [28]. For each case study in this paper the team members involved were around 10 to 16 but each case study was a part of larger project. We included this as teams were distributed and each project was a part of larger project. This kind of situation will allow us to study how collaboration and communication problems are handled in such cases. Figure 7 below shows number of project members involved in each of the case studies. The average number of members was around 93 taken from all 16 projects. The highest number of members was 210 in a project of Salesforce.com's R&D organization and it was divided into 30 Scrum teams working simultaneously in a single release code branch.

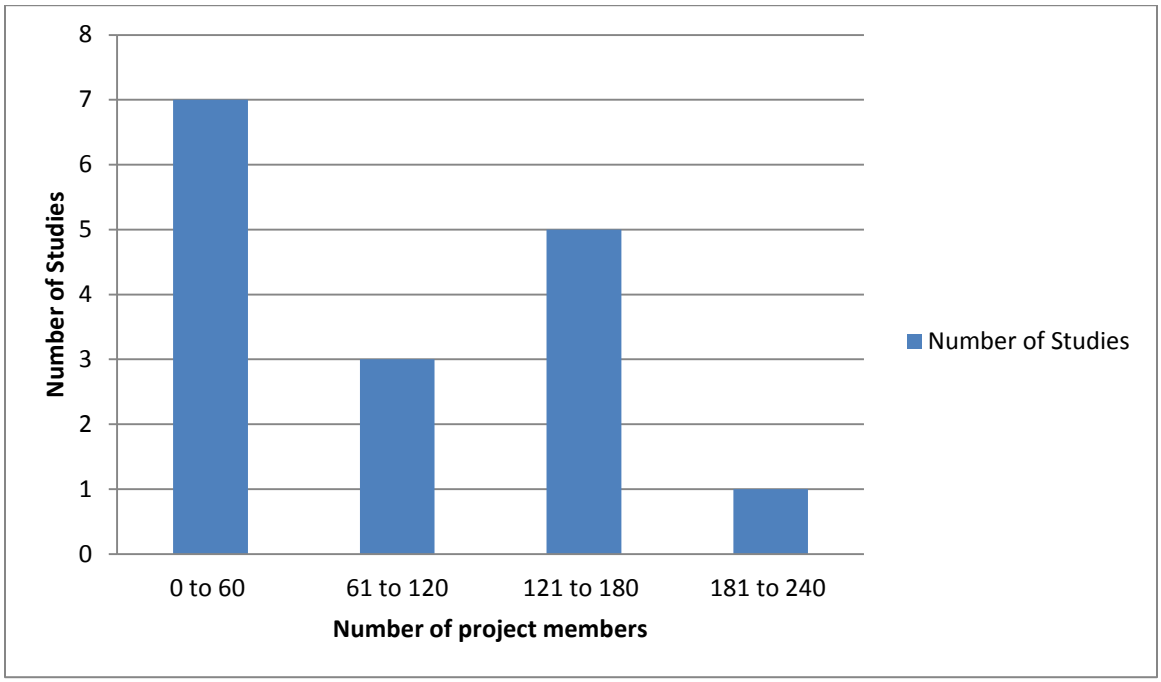

Figure 7 Classification based on number of project members

3.3.6. Number of Scrum Teams

Figure 8 shows the number of teams in each case study taken for this review. We observed a common pattern in most of these projects regarding the team size. Out of 16 projects reviewed, 11 projects had a team size ranging from 7 to 9. Other 5 projects had team size less than 7. From

this pattern we can conclude that even in large projects involving hundreds of personnel the ideal size of Scrum team should be 7 to 8 people per team. The average number of teams per projects is 13. Most of the teams in these case studies are distributed.

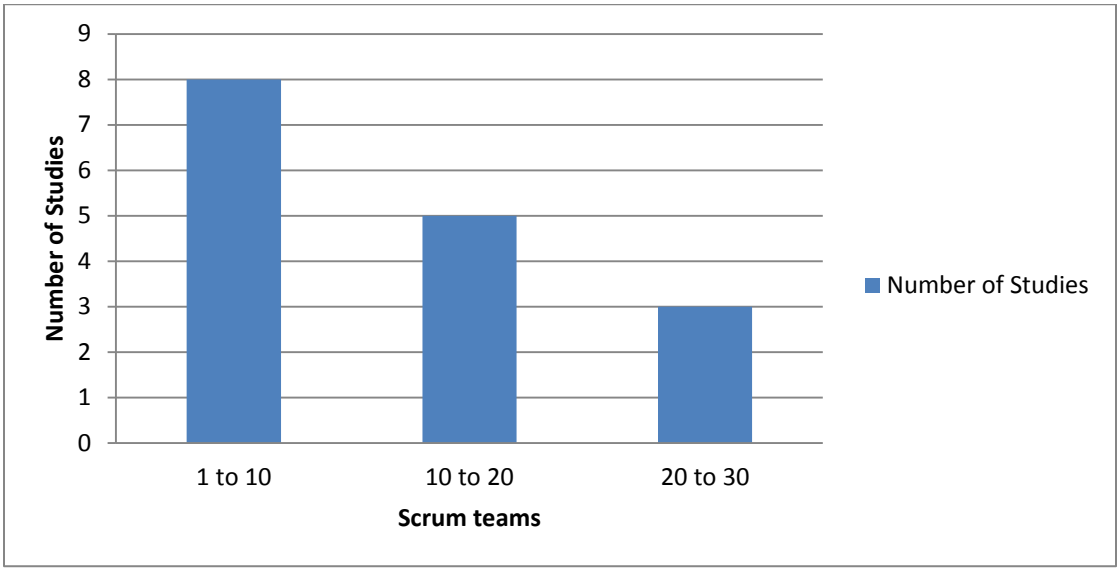

Figure 8 Classification based on number of Scrum teams

3.3.7. Practices identified to Scale Scrum

Through this SLR, we found that Scrum teams use various practices or strategies to support the use of Scrum practices in large scale projects. We extracted the data related to scaling practices from Table 2 and we formed a matrix of projects and their respective practices shown in Table 7.

Table 7 Matrix between Studies and their respective scaling practices

	SOS	APO'S	PPO'S	SOS Modified	Common Retrospective	Feature Backlog/Teams/SOS Meetings	Sprint Planning And Review
S1	X	X			X		X
S2(A)	X			X	X		X
S2(B)							X
S2(C)	X			X	X		X
S2(D)	X			X	X		X
S3					X	X	X
S4(A)	X						
S4(B)	X			X		X	
S5	X				X	X	

	SOS	APO'S	PPO'S	SOS Modified	Common Retrospective	Feature Backlog/Teams/SOS Meetings	Sprint Planning And Review
S6							X
S7(A)		X					
S7(B)			X				X
S8	X				X		X
S9	X						
S10	X				X	X	X
S11	X						X

SOS - Scrum of Scrums; APO - Area Product Owner; PPO –Product Proxy Owner

Each of the practice listed in table 7 is briefly discussed in Chapter 2. In this section we discuss how these practices are applied to real time projects and the challenges that have to be dealt with when applying them. We plotted a pie graph using data of Table 7. The pie graph in Figure 9 shows the adoption rate of each scaling practice used by the projects. It can be noted from this pie graph that SOS is the most commonly used practice along with Sprint Planning and review. In standard Scrum as well as Scaled Scrum, Sprint Planning and Review is mandatory and common. It has been observed that studies on scalability of Scrum reported mixed results for each of these practices. In the following few sections, each of this practice and their feedback are discussed individually.

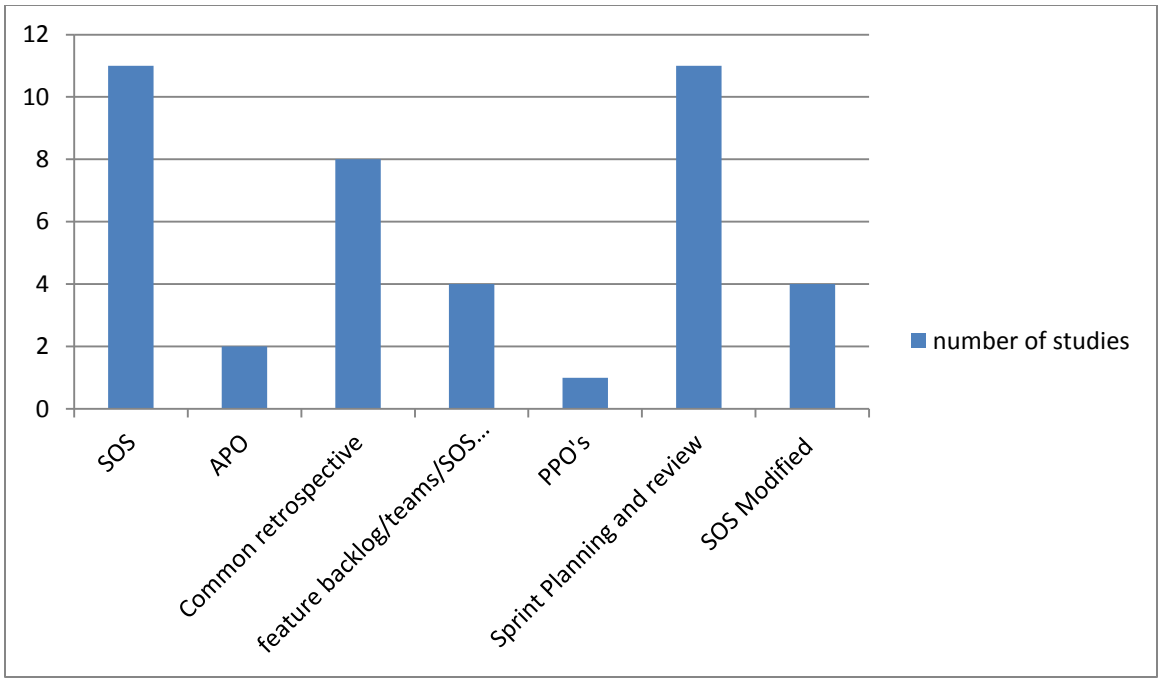

Figure 9 Practices used to scale Scrum Framework

3.3.8. Scrum of Scrums (SOS) Review

It has been observed that SOS technique is widely used to scale Scrum. Most of the studies included in this review have used Scrum of Scrums technique to develop the projects. However they have reported some positive as well as negative effects of using SOS. The division of reviews are shown in Figure 10 below.

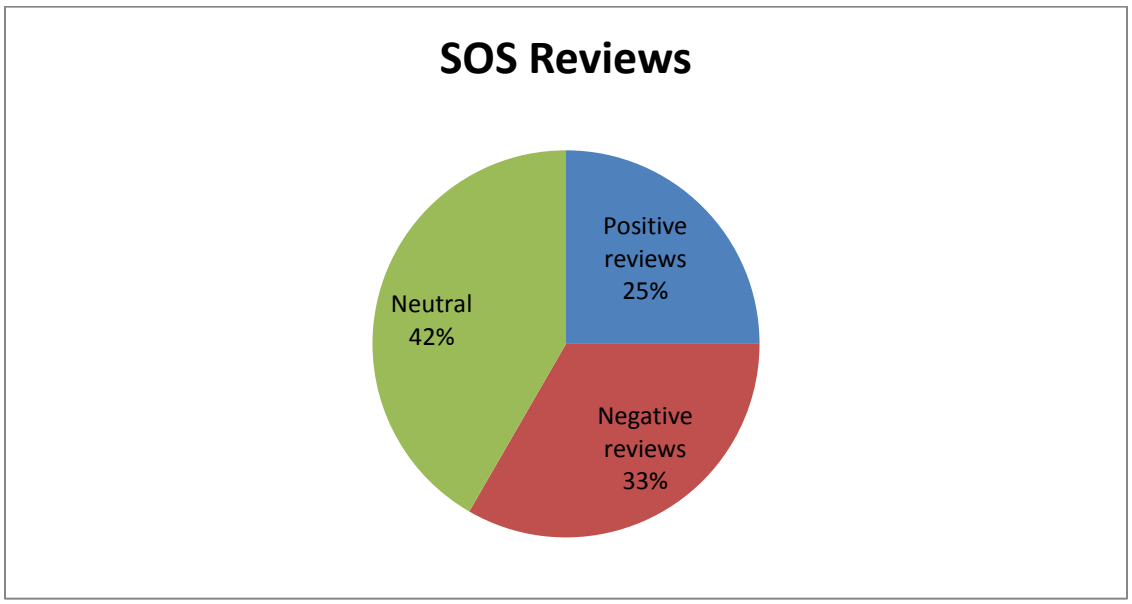

Figure 10 Reviews of SOS Practice

The study 1[42] discusses an ongoing case study on adopting and scaling Scrum in a large software development project distributed across four sites. In this case study along with SOS other scaling practices were also employed. The SOS meetings were conducted daily in which one representative from each team participated. The participant was responsible for reporting impediments experienced by the team. The challenge reported in conducting these SOS meetings was many teams were reluctant to report. Therefore, these SOS meetings in this case study were not useful and termed as unsuccessful by the developers.

The study 2 [28] presented four different case studies that investigate the impact of key project contextual factors on the use of Scrum practices in global software development. This study indicates that the team size had an impact on the Scrum of Scrums practice. For example, in case A and case B, the offshore teams operated as a single Scrum so the Scrum of Scrums practice was not used in that project. However, it was modified by holding weekly status meeting between onshore project manager and offshore team lead to resolve any cross team issues and dependencies. It was also noted that distance-specific factor such as temporal and geographical distance also impacted the Scrum of Scrums practice. In Case C, due to the high temporal distance, representatives from each sub-team used the mechanism *adjust working hours* to participate in a weekly status meeting. Similarly, in Case D, due to the geographical distance, Scrum masters and the project manager participated twice weekly in a status meeting via a teleconferencing. It was reported that SOS meetings improved team collaboration and created an environment of high trust. However, it was observed that effective use of Scrum meetings depends on good collaboration tools and other supporting mechanisms (e.g. adjust working hours).

In study 4 [46], a multiple case study is presented to show how Scrum-of-Scrum meetings were applied in two large-scale, globally distributed Scrum projects employing at least 20 Scrum teams. The results show that the Scrum-of-Scrum meetings involving representatives from all teams were not useful as the audience was too wide to keep everybody involved and the participants were reluctant to report thinking it might not be valuable to others. This often ended up with not reporting anything. Towards a solution to this problem, one of the case projects introduced feature-specific Scrum-of-Scrums meetings for 3-5 teams working on the same feature, which turned out to work well.

The study 5 [39] showed successful adoption of Scrum of Scrums by scaling team to multiple feature oriented teams. They demonstrated that separating teams by feature was perfect in order to focus on the main priorities for the development of the product and to avoid conflicts in backlog management. In addition to this they also pointed out that it is vital that every new team formed should have at least one professional with good technical skills and knowledge of Scrum, backed-up by an experienced ScrumMaster and a Product Owner.

Study 8 [37] employed Scrum of Scrums technique but did not report any positive or negative issues related with it. The study 9 [4] highlighted the practices that salesforce.com has been using successfully to scale Scrum and to manage inter-team dependencies. This project had over 30 Scrum teams working simultaneously in a single release code branch. Each business unit has a weekly Scrum-of-Scrums meeting and there is also a weekly Scrum-of-Scrum-of Scrums. Sometimes additional Scrum of Scrums was formed if there is a group of teams working closely together towards a common goal. They were using the 4 question format for the scrum-of-scrums and reported some redundancy between the team's Scrum-of-Scrums update and the team's status report. So they decided to use the open format Scrum-of-Scrums in which participants suggest discussion topics by writing them on the whiteboard at the beginning of the meeting. This approach made participants more responsible for the content of the meeting which resulted in more productive and collaborative discussions. Individual team status was not at all discussed in the Scrum-of-Scrums unless specifically raised as a discussion topic. However the status was always updated and available in weekly status report. They reported use of weekly status report as an important complement to Scrum of Scrums. They also pointed out that it is a challenge to deal with dependency management and cross-team coordination as the number of teams grows.

In study 10[43], Paasivara et al reports a case study on agile practices in a 40 person development organization distributed between Norway and Malaysia. They described how Scrum practices were successfully applied like daily scrum meetings, synchronized 4-week sprints and weekly Scrum-of-Scrums. They identified additional supporting practices for distributed projects such as frequent visits, unofficial distributed meetings and annual gatherings. In this project, a weekly Scrum of Scrums meeting was held in which one team member from each team participated. It is up to the team to select the participant and it is not mandatory that every time same member is selected. In addition to this all Scrum Masters also participate in this

meeting which is half an hour in length. Each team representative tells what his or her team has been doing since the last meeting, what it plans to do before the next meeting and what kind of impediments they have. Moreover, they added two additional questions: "Have you put some impediments in the other team's way?" and "Do you plan to put any impediments in the other team's way?" The goal of these questions was to ensure proper integration. The overall experience of using Scrum of Scrums practice reported in this paper was positive as it improved quality and ensured better and frequent communication.

Study 11 [5] reports the results of an interview study of five agile development projects by a single organization. Basic Scrum practice was used in these projects and specific details on adoption of SOS practice were not reported. However authors suggested that it is better to have some members of distributed teams to be collocated with Scrum Masters and Product owners regularly. They also suggested that well prepared global Scrum team meetings improve shared understanding and team coordination thus reducing integration problems.

3.3.9. Use of Area product Owners (APO'S) and Product Proxy owners (PPO'S)

Two studies have used Area product owners for scaling Scrum. In study 1[42], Paasivara et al used Area product owners to scale Scrum. In addition to the main Product owner role, the project had a team of APOs. Each APO and their respective team were responsible for individual features in the project. The role of APO was handled by two persons: a system architect and a product management representative. The system architect worked along with the team and also communicated with the product management representative. There was no communication between product management representative and the team. The system architect was collocated with the team at the main site. Each product area had a couple of development teams, and each feature would be implemented by a specific team. In this project the role of APOs was very useful and appreciated.

In Study 7 [45], Paasivara et al again focused on scaling the role of product owner in two large globally distributed projects each with 20 or more Scrum teams. Case A of this study used APO to scale Product owner role. APO's role was same as described in study 1 [42] above. They

suggested that APOs should be close to the development team and should conduct frequent workshops which improve team-APO communication. Case B of this study had a distributed project ownership team consisting of a Product owner (PO) and ten Product Proxy owners (PPOs). A large feature can be handled by 3 full time PPOs while a group of small features can be handled by a single PPO. The idea behind this was to have whole team responsible for all features. They reported that it is useful to have PO team located closer to the development team. The PPO was responsible for participating in daily scrums, arranging backlog grooming session, arranging face to face Sprint Planning, and also participating in team demos.

3.3.10. Feature Backlog/Teams/SOS Meetings

In Study 4 [46], Case B had Feature SOS meetings to solve problem of common SOS meetings. These meetings were held by 3-5 teams working on the same features. These meetings were held weekly once. However, the project sill had an integrated SOS meeting once a week and called it as the Grande SOS meeting.

These Feature SOS meetings were reported as useful by interviewees consulted in case study. The reason behind their success was people with common interests and goals shared their ideas and problems together. Grande SOS meetings gave the same negative results as SOS meetings in other case studies gave. Diverse and least interest of participants attending Grande SOS made it a failure.

In study 5 [39], multiple feature-oriented teams were used to scale scrum for large project. In this project special focus was given to the team's values such as commitment, teamwork and transparency. The 49 member team was divided along with backlog creating feature teams working on specific feature backlog. Each team was formed in such a way that it had at least three experienced Scrum people. Although the backlog was broken into two or more feature teams, these teams were well collaborated without intra team competitions. It is demonstrated through this study that dividing teams by feature was the best choice as it was easy to handle core priorities of the product and to avoid conflicts in backlog management.

Study 10 [43] described a project in which teams had their own backlogs. First the product backlog was divided among teams. The product owner of each team was responsible for updating backlog. The teams used a tool called Jira for managing backlogs. In addition to

individual team's backlog, the maintenance team has its own backlog where all product owners can add fresh issues. These issues are assigned a priority and addressed according to their criticality. This Jira tool was termed as satisfactory by all the team members. The usage of divided backlog in this project proved to be beneficial by improving communication and collaboration between all the team members.

3.3.11. Common Retrospective

Retrospectives are undertaken after the Sprint review by the Scrum team in order to inspect completed work and plan for improvement in the upcoming Sprint. In study 1 [42], Common retrospectives were tried using different ways. In the beginning, each team had their team specific retrospective first and then all teams gathered for a common retrospective. The problems were created as the team size grew larger and impediments remain unsolved. Therefore team members reported common retrospectives to be useless. Another way adopted was to conduct retrospective through an open space, i.e. anyone could suggest a discussion topic. At a time, several discussions were held and team members reported it as fun rather than a solution to the problems. Lastly they came up with a new format of common retrospective. The meeting was held and facilitated by an internal coach, who tried to avoid the earlier mistakes. The participants were asked to brainstorm the biggest impediments, prioritized them, pick the most important one, search the cause for it, select one root cause for which they brainstorm solution. Then they choose one solution for implementation. In the next common retrospective they would follow up on the implementation of the solution and probably work on the same impediment until it is fully solved.

In all the three case studies A, C, and D of study 2 [28], common retrospective was used. In case A, Retrospectives were held in the beginning 5 to 6 Sprints. However, the practice was withdrawn as the Scrum model was working effectively and any impediments was easily handled in other meetings. In case study C, Retrospectives were tailored to compensate the temporal distance between sites and the division of work. Two separate retrospectives were held each for onshore and offshore teams. In the retrospective, teams discussed successful strategies leading to work completion, impediments that were encountered and any improvements required. The results of each site's retrospective were posted in the project wiki, which was accessible by all stakeholders. In the beginning, retrospectives were held at the end of every Sprint. Later as

things were working smoothly, the time interval was increased to the end of every second Sprint. In study D, due to the temporal and geographical distances involved, Scrum team members used the mechanisms such as adjust working hours and teleconference to conduct their retrospective meetings. They proved to be successful in solving impediments.

In study 3 [34], Retrospectives meetings were held at the end of every iterations and project members seems to be content with the retrospective meetings as they gain valuable insights through them. In study 5[39], Retrospectives were initially conducted between cross functional teams and there was a failure due to disagreement among different teams. Then retrospectives were conducted between feature teams and they deemed to be successful. In study 8 [37] and study 10 [43], retrospectives were held in usual manner and reported to be successful.

3.3.12. Sprint Planning and Review

Sprint Planning and Review is inherent to the Scrum framework and used almost in all the case studies. However team members reported these meeting to be tedious and less beneficial to them whereas managers termed them as helpful in planning the upcoming Sprints. The common challenges reported by all the case studies which adopted Sprint planning and Review are:

1. When team members involved in Sprint planning have diverse interest, they hardly participated in the meetings. Even if they participated they showed no interest in solving impediments.
2. Moreover if teams are distributed then due to offshore team's lack of domain knowledge, the Sprint planning meeting becomes a long time consuming process that involves additional meetings.
3. Face to face Sprint Planning meetings proved to be beneficial but have to be tailored when teams and team members increases.
4. Distributed teams face communication problems due to poor network connections. Misunderstandings were common between distributed teams.
5. Sprint Planning meetings showed substantial improvement when someone guided them i.e. taking responsibility of the discussion.

However, in all the case studies Sprint Planning and review was used by tailoring it according to the project and team's situation.

3.4. Discussion

This section discusses how the data extracted from the different studies answered our research questions. The aim is to provide a synthesized overview of the current available literature addressing the issue of Scrum's scalability.

RQ1: Why to scale Scrum when used for large projects?

Larger projects have huge set of requirements which needs a huge workforce. Scrum framework was originally developed for smaller projects which have around 7 to 10 members. However when the same Scrum framework has to be used for larger projects it has to be scaled and sometimes tailored.

RQ2: What are the scaling practices currently used by the industry?

The scaling practices identified through this review are listed below:

1. Scrum of Scrums Review
2. Use of Area Product Owners
3. Use of Proxy Product Owners
4. Tailoring Scrum of Scrums
5. Use of Feature backlog
6. Use of Feature teams
7. Meetings held between feature teams
8. Sprint Planning and review

Each of this practice is discussed in detail in the preceding sections. Each practice has its own advantages and disadvantages. These practices are tailored according to the project's situation i.e. distributed or co-located, project's size, and team's convenience.

RQ3: Which Scaling practice is widely used and Why?

It has been observed that Scrum of Scrum technique is widely used to scale the Scrum framework for large projects. It is used in 8 studies out of 11 selected for this SLR. In this approach each Scrum team proceeds as normal, but each team has a representative who attends the Scrum of Scrums meeting to coordinate the work of multiple Scrum teams. These meetings are somewhat similar to daily Scrum meetings but they are not held every day. They are held

once every two weeks or three weeks. The factors leading to successful adoption of Scrum of Scrum (SOS) technique for scaling Scrum is summarized below:

1. SOS meetings improved team collaboration and create an environment of high trust.
2. Additional SOS meetings can be easily incorporated if there is a group of teams working closely towards a common goal. It is easy to manage inter team dependencies in such cases.
3. In study 10 [43], SOS improved quality, ensured better and frequent communication through frequent visits, unofficial distributed meetings and annual gatherings.
4. Feature specific SOS meetings for teams working on the same feature performed very well in all the case studies in this SLR.
5. The SOS meetings can be scaled up in a recursive manner.

After careful review of available literature, it can be concluded that SOS technique is used widely in current industry and it is easy to adapt. In some projects when team size grows, it might create some problems which can be solved with the use of feature specific teams.

RQ4: What changes are required to the original Scrum framework to make it adaptable to large projects and what effect does that have on quality and productivity?

In particular, no changes are required to the original Scrum framework to use it for large projects. In fact following the Scrum guidelines and principles makes it more effective to be used for large projects. The Scrum framework was initially used for the small project involving 5 to 10 members. It is said that to be effective Scrum teams should have an ideal size of 7 to 10 members. This notion may be the cause for the fallacy that the Scrum framework can only be used for small projects. However, it can easily be scaled for effective use in larger projects. In situation where the team's size exceeds 10 people, multiple Scrum teams can be formed to work on the project. The Scrum of Scrum technique is the best technique so far to facilitate coordination among Scrum teams. Multiple Scrum teams work in parallel and they have to be synchronized through flow of information and frequent communication. The SOS meetings greatly help to keep the teams synchronized. There are no specific rules defined for the frequency of SOS meetings. It can be adjusted based on amount of team dependency, project's size, geographical distance between teams and level of complexity.

Only five studies out of 11 studies reported the effect of scaling Scrum on Productivity and quality. All of them reported a positive result about the Scrum being used for developing large projects. Study 1 [42] reported that adoption of Scrum resulted in quick releases for customers to test. S3 [34], S5 [39] and S6 [23] reported an increase in productivity and quality. The factors leading to increased productivity were listed as increased team velocity, teams with mixed high skills, anchored methodology, improved motivation, business and user involvement, colocation, good infrastructure and frequent communication.

RQ5: What are the challenges faced by developers in adopting Scrum for large projects?

The challenges while adopting scrum for large projects are:

1. Ensuring good communication between the team members and also between the teams for a large project. Everyone involved in the project should know the Status of the project.

2. The major challenge as pointed out by inventor of Scrum Jeff Sutherland is it is difficult to change the mindset in the organization in general and on management-level in particular [57]. Team learns agile practices and adopt, but there is a high chance the team again uses the traditional practices, because of which again productivity degrades.

3. High performing teams rely heavily on open communication, feedback, and motivation. For a large team it will be difficult to monitor teams.

Through this SLR, it has been observed that the biggest challenge faced in Scaled Scrum is the assurances of perfect inter team and intra team communication. Due to large team sizes it is difficult to completely eliminate this communication overhead. However this overhead should be monitored to control the project's performance.

Chapter 4

System Dynamics Modeling

This chapter presents the second phase of the thesis which involves construction of causal loop diagrams and a simulation model for Large Scale Scrum. In this part of the thesis we adopt system dynamics approach to realize a model for analyzing the dynamic behavior of the large scale Scrum. The results obtained from systematic literature review described in chapter 3 are used to construct these models. In this chapter we first provide an introduction to system dynamics so that different concepts used in modeling can be easily understood.

5.1.Overview of Modeling and Simulation

To understand Software process simulation, some important concepts related to modeling should be understood first. In this section we provide brief concepts related to modeling and simulation.

According to Kellner et al [30]:

1. A model is an abstraction of a real or conceptual complex system. A model is designed to display significant features and characteristics of the system which one wishes to study, predict, modify or control. Thus a model includes some, but not all, aspects of the system being modeled. It provides valuable insights, predictions and answers to the questions it is used to address.
2. A simulation model is a computerized model which possesses the characteristics described above and that represents some dynamic system or phenomenon.
3. A software process simulation model focuses on some particular software development/maintenance/evolution. Since all models are abstractions, a model represents only some of the many aspects of a software process that potentially could be modeled namely the ones believed by the model developer to be especially relevant to the issues and questions the model is used to address.

Kellner et al [30] categorized reasons for using simulations of software processes as: strategic management; planning; control and operational management; process improvement and technology adoption; understanding; and training and learning. Software development is a

complicated process and requires lot of resources which includes workplace, people, software resources and documentation. Thus experimentation with software development requires use of these resources and any misuse of these resources in reality poses a risk in terms of cost as well as time. In such cases simulation is the best option for any researcher. It allows simulating the behaviour of the system under different conditions without actually exploiting the resources and also saves time. Software development process simulation models are used to capture dynamic relationships and behavior inherent in software development projects as well as process level concerns.

Two main modeling approaches used for software process simulation are system dynamics (continuous simulation) and discrete event simulation [39]. System dynamics models [39] describe the interaction between project factors, but do not easily represent queues and discrete process steps. System dynamics models describe the system in terms of 'flows' that accumulate in various 'levels'. The flows can be dynamic functions or can be the result of other 'auxiliary' variables. As the simulation progresses in small evenly spaced time increments, it computes the changes in levels and flow rates. Discrete event models describe process steps, but may not have enough events to represent feedback loops accurately [39]. Discrete models are often used to model a manufacturing line where items or 'entities' move from station to station and have processing done at each station. According to Kellner et al [30], each of these techniques has their advantages and disadvantages as shown in Table 8.

Table 8 Advantages/Disadvantages of system dynamics and discrete event Approach adapted from Kellner et al [30]

	System Dynamics	Discrete Event
Advantages	Clear representation of the relationships between dynamic variables Accurately captures the effects of feedback	Queues and interdependence capture resource constraints Attributes allow entities to vary CPU efficient because time advances at events
Disadvantages	No ability to represent entities or attributes Sequential activities are more difficult to represent	No mechanisms for states Continuously changing variables not modeled accurately

5.2.Introduction to System Dynamics

System Dynamics (SD) was introduced by Dr. Jay W. Forrester from Massachusetts Institute of Technology in 1960 to apply engineering principles to social systems [19]. It is used for modeling and understanding the dynamic behavior of complex systems. The models are built using cause and effect relationships with the help of causal loop diagrams, Stock-flow diagrams

with levels and equations. The equations govern system behavior. Abdel-Hamid and Stuart Madnick [1] first used system dynamics for software process simulation. Madachy [38] modeled a more detailed development process. Abdel-Hamid and Madnick's model [1] was able to reproduce several predictable project characteristics, suggesting that much of project behavior was a consequence of relationships between factors.

5.2.1. Building a System Dynamics model

System dynamic models are constructed based on the principle of cause and effect, delay, and feedback. The basic idea is that actions and reactions have consequences, for example quality affects sales, and extra staff affects the delivery time. These kinds of cause and effect relationships are simple but when they are combined by several other long chains of dependent factors, they can become sophisticated. Feedback is the process which shows how a change in one factor affects another factor. A feedback loop is a closed sequence of causes and effects, a closed path of action and reaction. A feedback system is formed from the interconnection of feedback loops.

In this thesis, we used Vensim PLE [62] software to develop the simulation model. It is a software program that facilitates the development, exploration, analysis and optimization of system dynamic models.

Causal loop diagrams are often used in system dynamics to illustrate cause and effect relationships. In causal loop diagrams arrows represent the relationships between various variables. An arrow marked positive or *s* indicates that both variables change in the same direction i.e. when one increases, the other increases. An arrow marked negative or o indicates that both variables change in opposite directions i.e. when one increases, the other decreases and vice versa.

The commonly used constructs in SD models are a level, a rate, an auxiliary variable, a source and a sink [38]. The definitions of each of them are described below and graphical notations are presented in Figure 11.

A *Level*, also called a Stock, represents an element that accumulates or drains over time, e.g. Work to be done, developed software.

A *Rate*, also called a Flow, represents an action that changes the value of levels over time, e.g. software development rate, assimilation rate etc. The value of a rate is not dependent on previous values of that rate; instead the levels in a system, along with other auxiliary variables.

An *Auxiliary variable* assists in adding details to level and rate elements, e.g. communication overhead %, or simply constants, e.g. average meeting time.

A *Source/Sink* is a point of reference for communication with systems or processes outside the scope of system being developed, e.g. software delivered to customers.

A *link* is used to represent a dependency between two elements. These are information linkages which represents information flow between different elements of SD model.

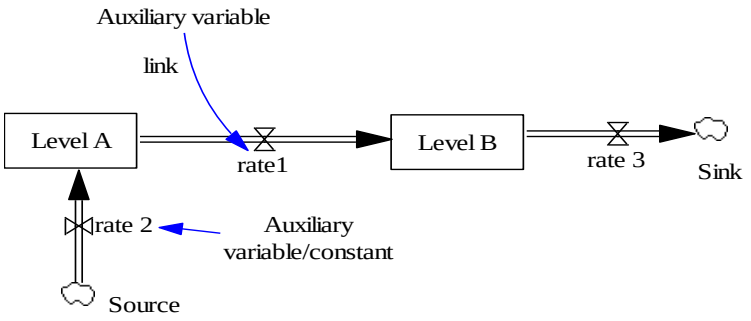

Figure 11 Elements of System dynamic model

5.3.Modeling the Dynamics of the Scrum Framework

For the development of SD model for the Scrum framework, we adopted the four stages of modeling described in [48].The four stages of modeling are outlined in Table 9, with the detailed steps of each stage listed under them. We elaborate each modeling stage in detail in the following sections.

1. Conceptualization • Define the purpose of the model • Define the model boundary and identify key variables • Diagram the basic mechanisms, the feedback loops, of the system
2. Formulation • Convert feedback diagrams to level and rate equations • Estimate and select parameter values
3. Testing

• Simulate the model and test the dynamic hypothesis • Test the model's assumptions • Test model behavior and sensitivity to perturbations
4. Implementation • Test the model's response to different policies • Translate study insights to an accessible form

Table 9 Stages of SD Modeling adapted from [48]

4.3.1. Conceptualization

The main objective of this thesis is to investigate management challenges in adopting Scrum for large scale projects. From the first phase of this research i.e. through Systematic Literature review, we identified several scaling practices used to scale Scrum for large projects. We outlined advantages and disadvantages of each of them reported in literature. Among them the most widely used techniques were Scrum of Scrums (SOS) and Sprint planning with review. It has been observed that Sprint planning with review is already an inherent step in the Scrum framework which had to be carried out for all kind of projects. SOS is specifically used in larger projects involving large number of members. Therefore we selected SOS technique to study in detail by reflecting on its dynamics during software development process. The details pertaining to operation of SOS technique is discussed in Chapter 2.

The purpose of this model is to examine the relationships between various factors affecting the performance of the Scrum framework as well as Scaled Scrum. These factors/variables were identified in the Systematic literature review.

The exogenous inputs to our model are project-specific parameters (project backlog, staff, number of Sprints, etc.) and project-team specific parameters (productivity, rework discovery rate, sprint velocity, etc.). Based on such inputs, model simulations capture project performance as output in terms of cost, schedule, and quality.

Based on results obtained from Systematic literature review, we devised causal loop diagram for the basic Scrum framework and also for the scaled Scrum framework. In this model we differentiate the basic Scrum and SOS by varying the number of staff and in some cases project size. We devise two different causal loop diagrams for Scrum and SOS to highlight explicit factors which has to be considered while scaling Scrum in SOS. All the variables involved in the basic Scrum process are also applicable in SOS with some additional variables.

The following set of causal loop diagrams are developed as a basis for the construction of SD model. These loops include several different relationships that were identified from the available literature and software development experience. Only a subset of these loops is captured in the detailed SD model.

Figure 12 shows the causal loop diagram for the Scrum framework. It is simply a representation of the relationships between the system components that create the dynamic behavior of the Scrum process. Most of the variables in Figure 11 have been already discussed in detail in Chapter 2. However we provide a short description of these causal loop diagrams and variables involved.

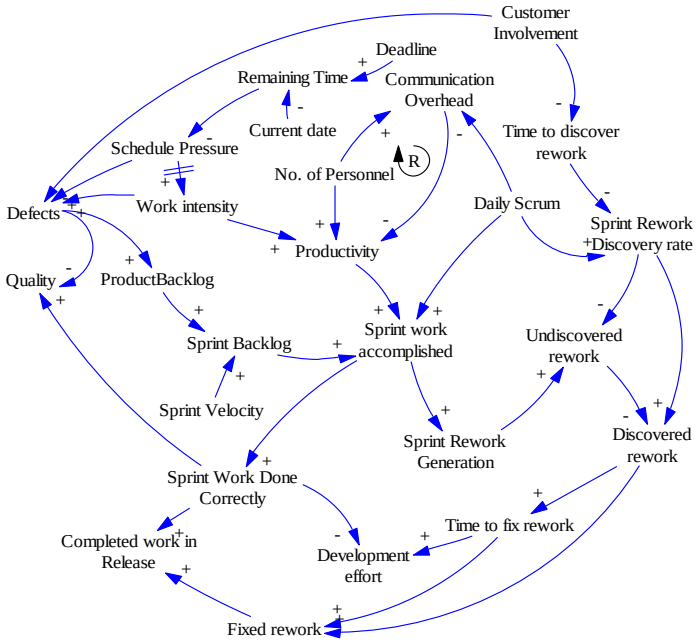

Figure 12 Causal loop diagram for the Scrum framework

Product Backlog: The amount of work to be done increases with the increase in requirements and with any defects detected.

Sprint Backlog: It is the work to be accomplished in each Sprint and increases with an increase in Sprint velocity and Product backlog.

Sprint work Accomplished: It depends on the productivity of the team. It increases with an increase in productivity and enhanced through daily scrum meetings.

Sprint rework generation rate: This increases with the increase in Sprint work Accomplished and causes the reduction on amount of undiscovered rework.

Schedule Pressure: As the *Deadline* approaches, *Schedule pressure* on the staff increases which in turn increases *work intensity*. At the same time this also results an increase in number of *defects*.

Productivity: There is reinforcing loop between *number of personnel* and *communication overhead* which effects productivity. Increasing the number of personnel results in increase in productivity but however communication overhead resulting from it reduces productivity.

Figure 13 shows the causal loop diagram which includes factors which are specific for large scale Scrum. All the factors from the basic Scrum causal loop diagram and the factors from Figure 12 affect the performance of Scaled Scrum.

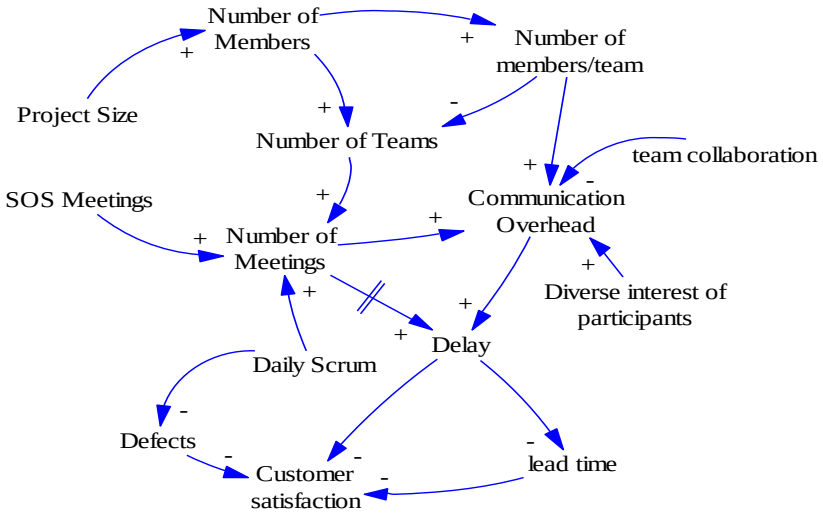

Figure 13 Causal loop diagram for the scaled Scrum

Project Size: Larger projects involves unprecedented amount of requirements, covers broader scope, and sometimes time critical. Development of such kind of projects requires large number of personnel.

Number of members/personnel: More *number of personnel* results in an increase in *number of teams* which in turn increases *number of meetings* and *daily Scrums*.

Communication Overhead: More number of personnel involved increase in number of communication pathways which results in communication overhead. Moreover *communication overhead* also increases if the participants involved have diverse interest. On the other hand, *Team collaboration* reduces communication overhead.

Delay: *Number of SOS meetings, Daily scrums*, and *communication overhead* increases the *delay* which effects the delivery time and hence results in *customer dissatisfaction*.

4.3.2. Formulation and Construction of SD Model

In this phase of study, we develop a simulation model based on the causal loop diagrams and data obtained from SLR. Formulation and construction of a model involves iterative elaboration, construction, and simulation events. The model is constructed iteratively by adding stocks, flows and variables at each step and testing the model to keep its growing elaboration under control. Figure 14 shows the first iteration structure of the model. Here we model the basic Scrum framework by considering its core features.

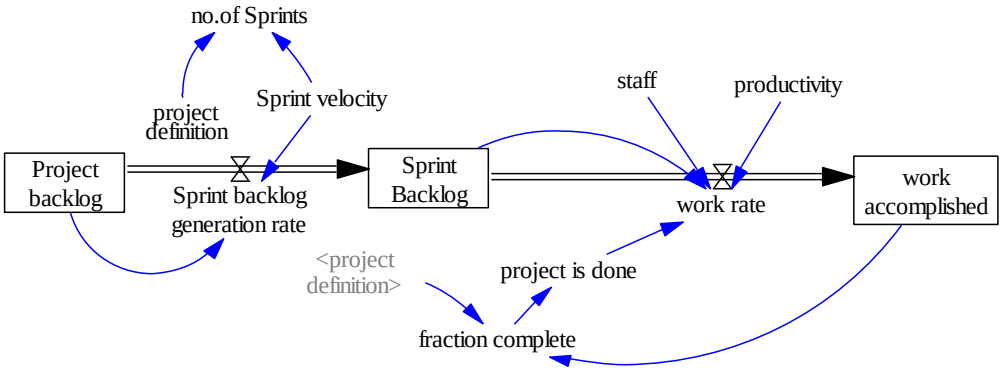

Figure 14 SD Model Iteration #1 Structure

The following stocks of work exist during software development using Scrum:

Product Backlog: A Product backlog is a set of requirements that are prioritized and it is the product road map.

Sprint Backlog: The list of tasks/work that the development team must address during the upcoming Sprint. It basically consists of set of tasks required to complete a feature.

A Project begins with an initial amount of work *(Project definition*) in the *Product Backlog*. A subset of those tasks is moved into the *Sprint backlog* at the start of each Sprint. This subset of tasks from product backlog is selected depending upon the Sprint velocity of the team. The requirements from *Sprint backlog* are developed depending upon the productivity of the available staff and the accomplished work is accumulated into the stock *Work Accomplished*.

All the backlogs are modeled in terms of fungible tasks as their unit. In this first iteration, we assumed the perfect development of software ignoring the dynamics of the rework cycle. It means work is accomplished without any errors/rework and accumulates in the stock *Work Accomplished*. In this first iteration we have set the following initial input parameters for the sample project:

Project definition/size: 500 tasks

Sprint velocity: 25

Staff: 10

Productivity: 0.4

For initial testing of the work rate, we assume an average value of 0.4 tasks / person-day. This value is selected based on published data sources and other models. The work rate will be:

Work rate = (0.4 tasks / person-day) · (10 people) = 4 tasks/day

With this equation, the work rate will be constant throughout the simulation, the developed software will rise linearly, and the project completion time can be calculated as:

Project Completion time = (500 tasks)/ (4 tasks/day) = 125 days.

The simulation is run and checked against the above calculation and result is shown in Figure 15. To determine exactly the project completion time we defined a variable called '*project is done'*. This is used as a Boolean variable and is set to 1 or zero depending upon the fraction of work remaining. In the model, we set it to be 1 when all work is done and 0 when even one task is remaining. This can be simply done by using IF THEN ELSE statement. Figure 16 shows that Project is completed at 125th day when *project is done* variable reaches 1.

Project is done = IF THEN ELSE (Fraction Complete >=1, 1, 0)

Where, Fraction Complete= work accomplished / project definition

Evolving this model into the next iteration by adding the concept of rework yields the second iteration of the model as shown in Figure 17, where a *Fraction Correct and Complete* (*fcc*) dictates the percentage of completed work that is correct and defect-free, ending up in the stock of *Work Accomplished*. The rest of the work is either incorrect or incomplete and requires rework, and thus accumulates into the *Undiscovered Rework* stock.

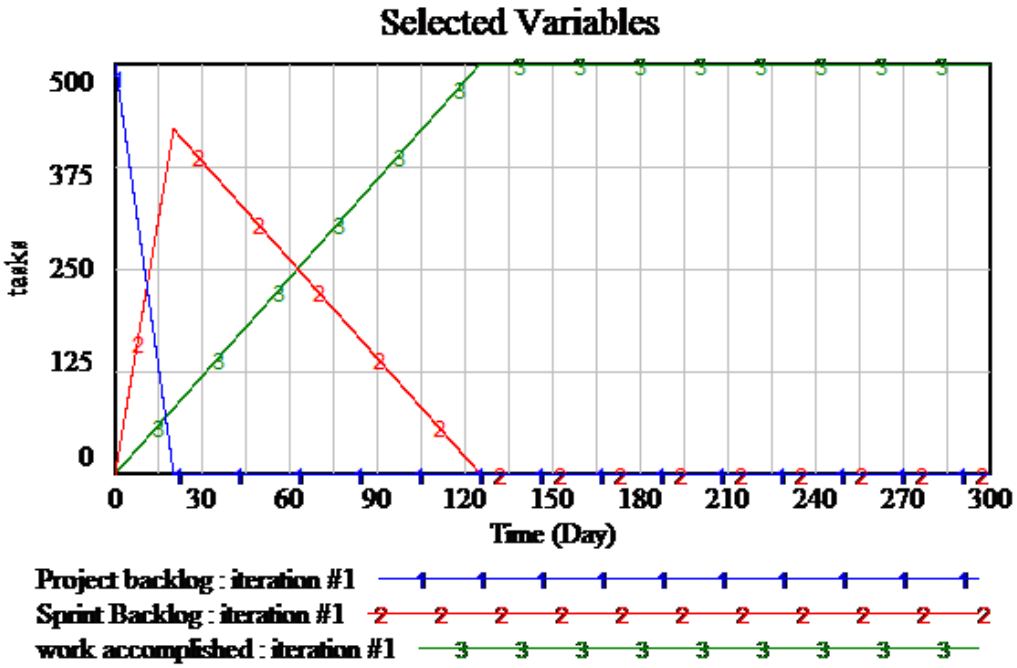

Figure 15 Work Transfer through backlogs in iteration 1

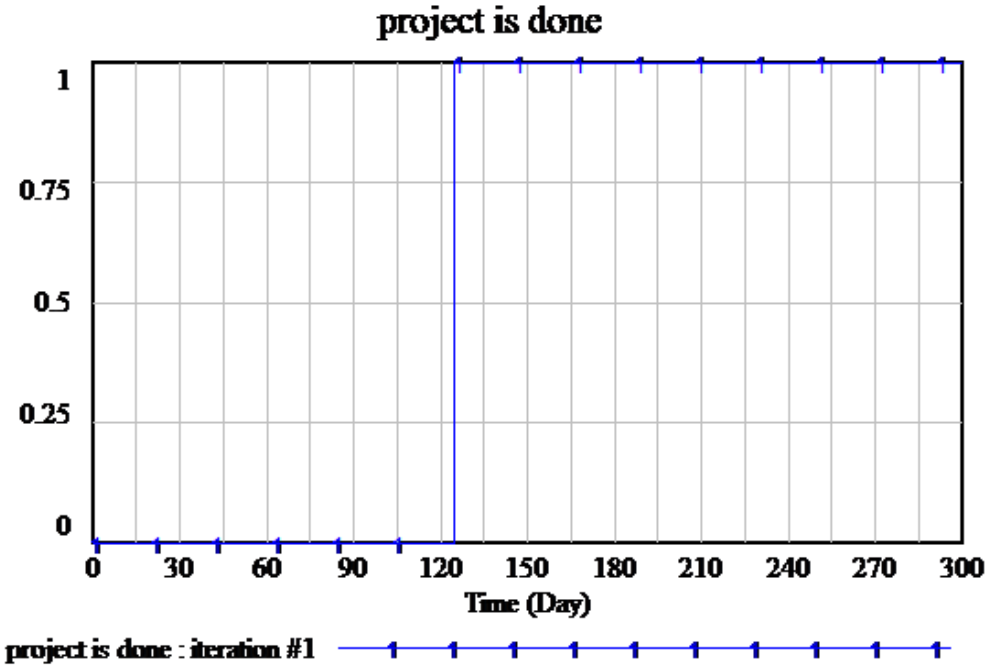

Figure 16 Graph showing Project completion time for iteration 1

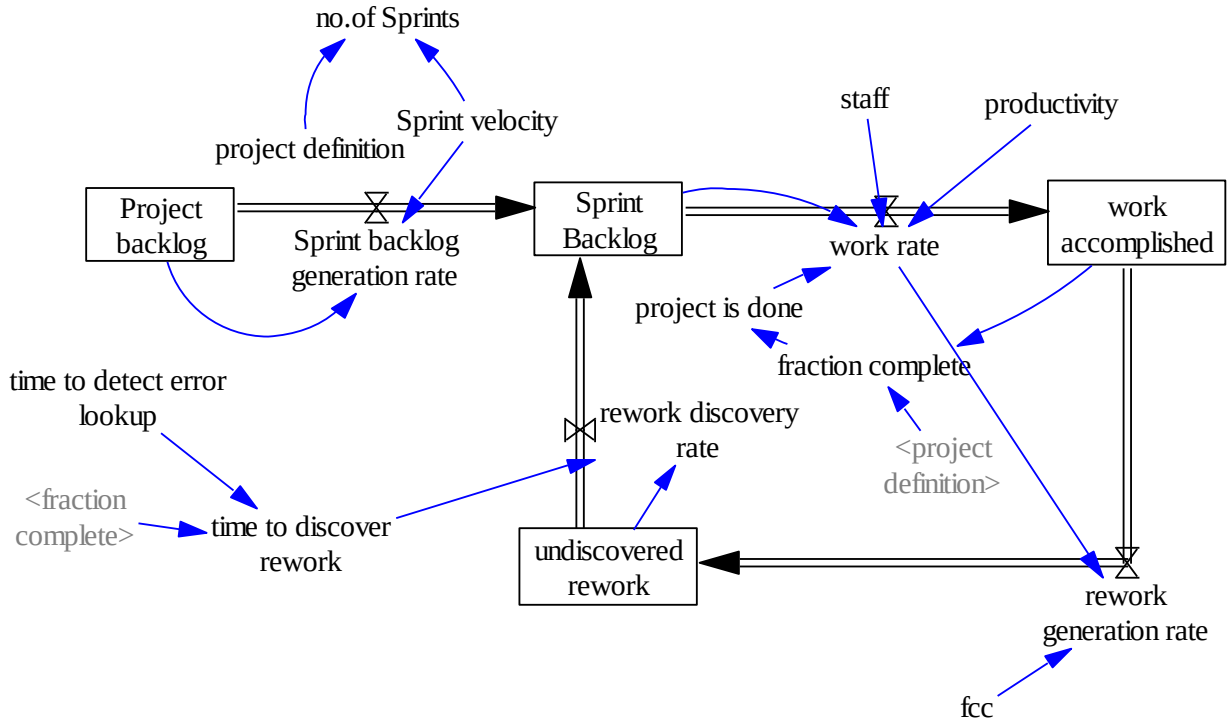

Figure 17 SD Model Iteration #2 Structure

Continuing with our sample project, assuming an *fcc* of 90%, It has been found that 50 tasks worth of rework have been introduced into the system by the 125th day, as shown in the graph in Figure 18, generated by executing this iteration of the model.

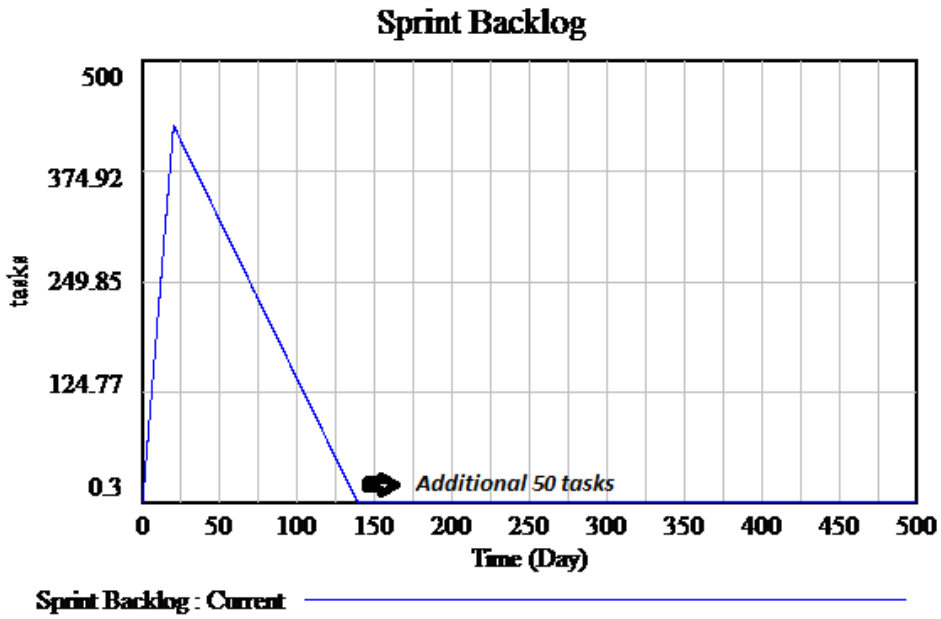

Figure 18 Graph showing additional work

The rework is discovered based on the variable *time to discover rework.* In real time projects, the rework is discovered at various stages so look up function is used to model *time to discover rework* depending upon the status of the project. To accomplish this, a look up function on *fraction complete* is used to drive *time to discover rework to* a much smaller value towards the end of the project.

Time to discover rework = time to detect error lookup (fraction complete)

Time to detect error lookup = [(0,0)-(10,10)],(0,5),(0.5,3),(1,0.5),(1.1,0)

The model is now simulated and the result is shown in Figure 19. After adding the rework cycle to the model, the work takes an additional 14 days for completion.

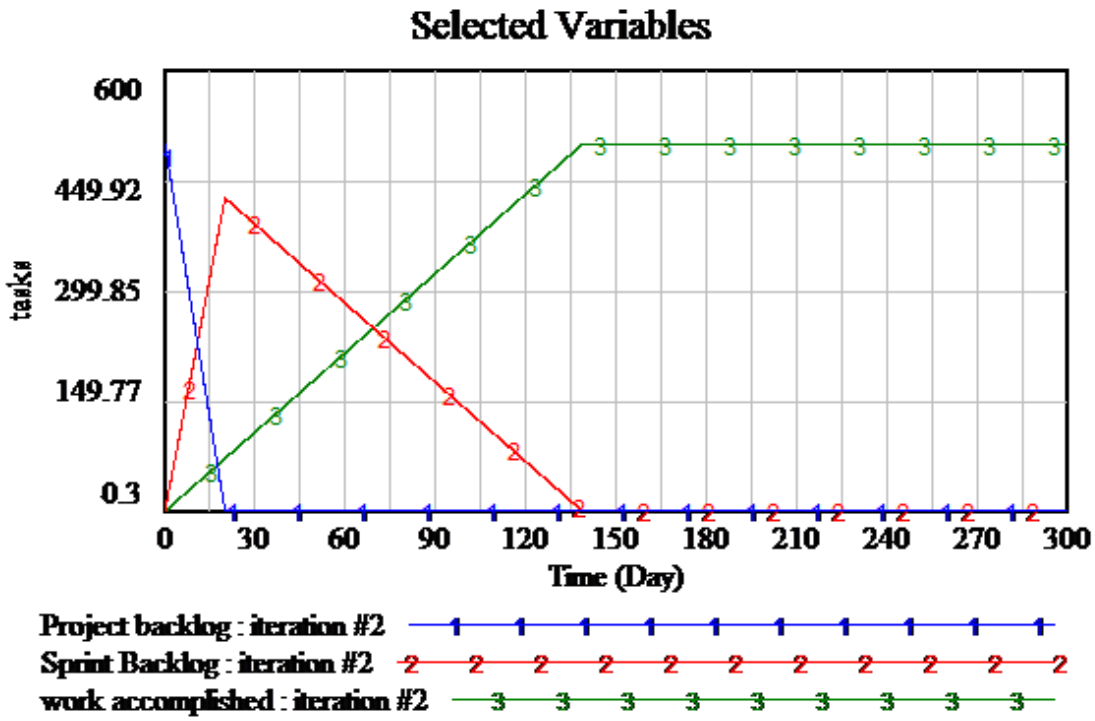

Figure 19 Work Transfer through backlogs in iteration 2

Due to the discovery of rework whose presence is evident in any kind of project, a project that was planned to complete in 125 days now takes up to 139 days as shown in the output graph in Figure 20. Therefore, the rework cycle is considered as a heart of the dynamics behind project performance. Undiscovered rework and defects leads to additional work in the form additional iterations/Sprints which are unplanned. It has been observed by simulating this model with varying inputs that project completion time can be reduced by improving fraction correct and complete (fcc) and discovering rework as soon as possible.

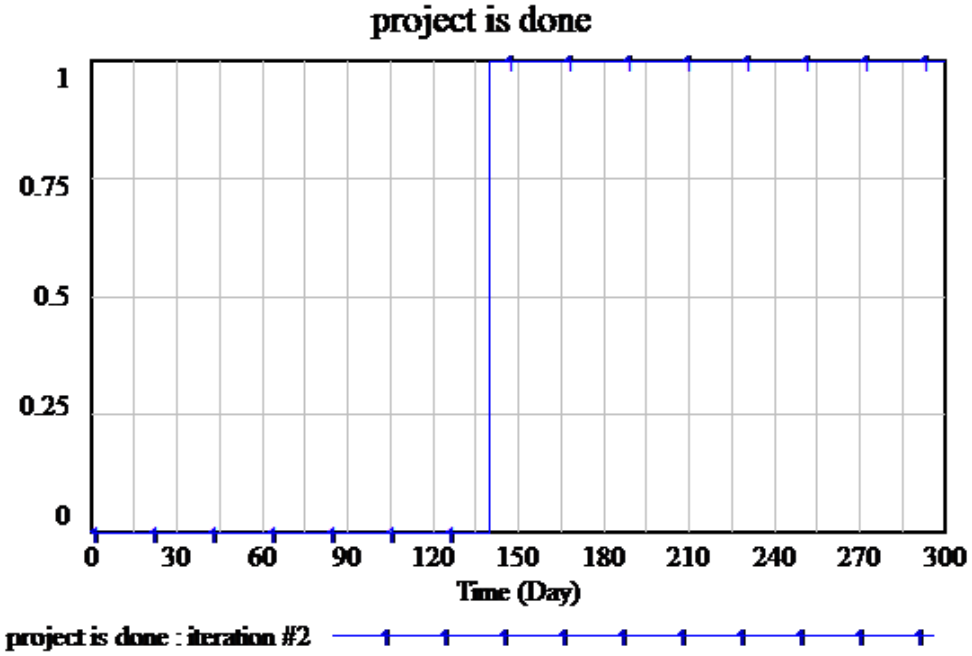

Figure 20 Graph showing Project completion time for iteration 2

In iteration 1 and 2 of the model, the experience of staff is assumed to be constant. Increasing the number of personnel in the model resulted in an increase in work rate and thereby project was completed sooner for the same value of productivity.

In the next and final iteration of the model, we consider the experience of the personnel involved in the project and also the communication overhead which resulted with an increase in project members. The main objective of the model is to investigate the effect of increased personnel on the project's performance. Through systematic literature review, we came across several papers which discussed communication overhead as the main barrier encountered while adopting Scrum for large projects. In this SD model, the Brooks's law is incorporated to study the relation between number of personnel and communication overhead in the Scrum framework.

Fred Brooks first articulated "Brooks's Law" in the book "The Mythical Man Month: Essays on Software Engineering" [9] as:

Adding manpower to a late software project makes it later.

Brooks's law can be generalized as:

"Adding people to software development slows it down"

The model is evolved to depict effort and time required to complete the project. It allows tracking the *work rate* over time and assessing the final completion time to develop the tasks in project backlog under varying conditions. The model developed represents a nominal case, and would require calibration to specific project environments.

As time progresses, the number of tasks in Product backlog and Sprint backlo*g* decreases since they represent user stories remaining to implement. These user stories are processed over time at the *work rate* and become *work accomplished,* so number of in *Sprint Backlog* decrease as *work accomplished rises*. The *work rate* is constrained by factors such as: the *nominal productivity* of a person, the *communication overhead %,* and the number of personnel.

The number of personnel is divided into new and experienced personnel and effective staff is total of new and experienced personnel minus the number of personnel providing training for the new staff. The *communication overhead %* is described as a non-linear function of the total number of personnel that need to communicate ($0.06n^2$, where *n* is the number of people) [1]. The *experienced staff needed for training* is the training overhead percentage as a fraction of a full-time equivalent experienced staff. A new person is trained by taking a quarter of experienced personnel time and the average rate for assimilation of new employee is 20 days.

There is a change in the regular behavior of the system when new personnel are added to the project. This result in following effects: 1) An increase in communication overhead, 2) An increase in training overhead, and 3) An increase number of personnel working on the project. When new person joins the project then they require training which will cost experienced personnel's time. Due to an increase in number of personnel, there is an increase in communication overhead. This communication and training overhead lead to decrease in productivity. But at the same time an increase in number of staff causes productivity to increase.

The nominal productivity is set to 0.4 tasks /person-day, with the productivities of new and experienced staff set to (0.8 * nominal productivity and 1.2 * nominal productivity) adapted from the COCOMO II cost model experience factors [8]. Figure 21 shows the SD model developed for

the Scrum framework. It can be noted that in this model only subset of variables are selected from causal loop diagram devised. This model can be used for small scale projects as well as large scale projects. This can be done by simply varying the input parameters such as project size, Sprint velocity, and number of new and experienced personnel to determine the project performance. The simulation results are discussed in the following result's section of the thesis.

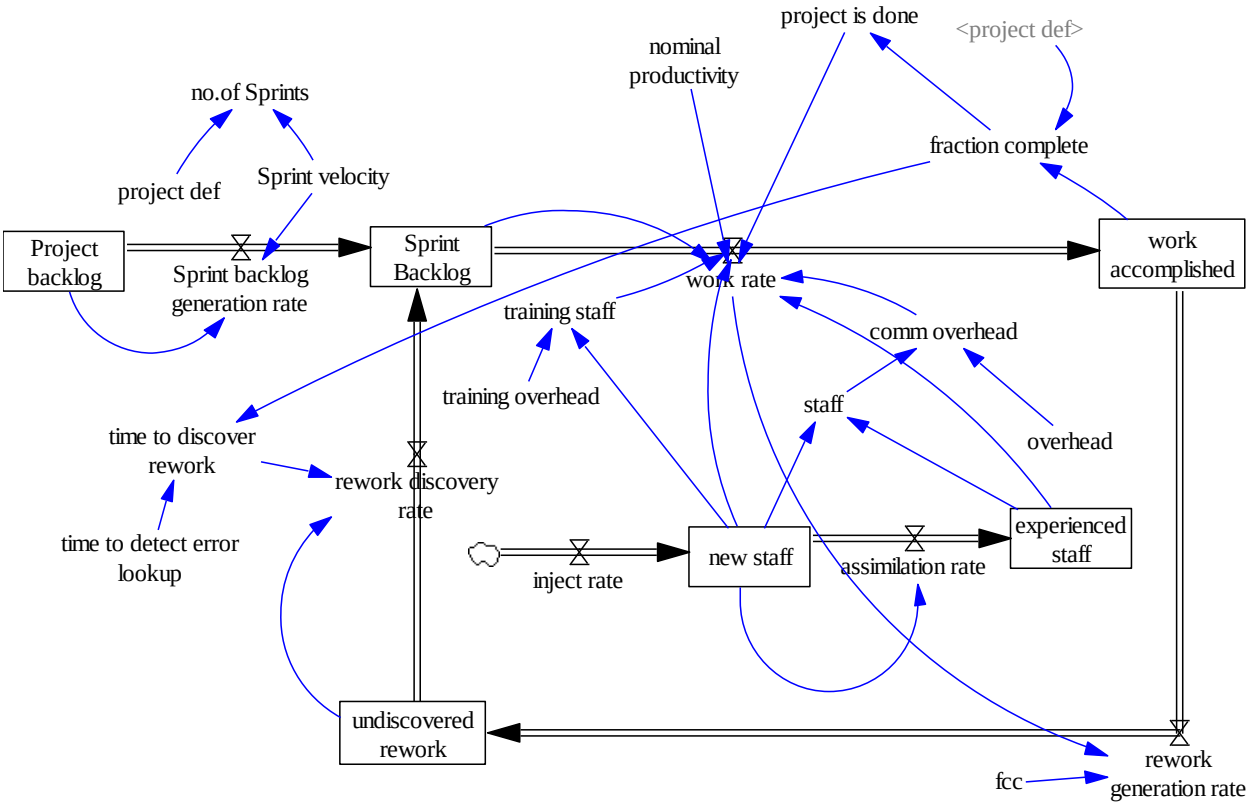

Figure 21 A System dynamics model for the Scrum framework.

4.3.3. Testing and Implementation

Testing of SD models involves model calibration and validation. Model calibration is the process of estimating the model parameters to find the similarity between observed and simulated structures and behaviors [41]. Model calibration is easy when the modeller have the access to data-sets from real projects. However when no such data sets are available then the issue can be resolved through judgemental estimation and using data-sets that have been previously made available online. In this thesis, some of the model elements were calibrated based on data collected from SLR, some were from developed system dynamics model [1] and remaining elements were calibrated using judgemental estimation. After the initial calibration the model

was validated using the tests described in the next section. Following that, the model was simulated and the results obtained were analyzed.

Forrester and Senge [20] described multiple tests which can be used to validate a System dynamics model. According to Forrester and Senge [20], there is no single test that can be used to validate a System dynamics model. The validation of System dynamics model is carried out using a wide range of tests, including tests of model structure and the ability of the model to mimic real-life behavior. The validation process is performed in two steps. The first step involved a structural assessment of adjacent elements, model subsystems, and overall model structure [9]. Once enough confidence in the structural validity of the model had been established, in the second step the behavioral validity of the system was tested to establish to relate the model behavior with the information gathered from SLR.

Sterman [59] described twelve tests identified in Table 10 that may be used for structural and behavioral validation. All the tests were performed except two that were not applicable to this research. Hence the model was successfully validated.

Table 10 Summary of Model tests (Adopted from [59])

Name of the test	**Purpose of the test**	**Procedures conducted in this research**	**Test Results**
1. Boundary Adequacy	Determines whether the important concepts are included in the model	Model causal diagrams and Stock-flow diagrams were reviewed using available SLR	Model was improved
2. Structure Assessment	Determines whether model structure is consistent with the relevant descriptive knowledge of the system	The major relationships, input variables and output variables are reviewed using available similar kind of models	Passed
3. Dimensional Consistency	Determines whether each equation is dimensionally consistent	Verified dimensional consistency using Vensim's inbuilt tool	Passed

Name of the test	Purpose of the test	Procedures conducted in this research	Test Results
4. Parameter Assessment	Determines whether Parameter values are consistent with relevant descriptive and numerical knowledge of system	Each parameter values are set based on data collected from SLR, some standard laws and judgemental estimation	Passed
5. Extreme Conditions	Each equation makes sense on extreme input values	1. Inspecting each equation 2. Testing response to extreme values of each input	Passed
6. Integration Error	Determine whether the results are sensitive to the time step	Used different time steps to ensure proper results	Passed
7. Behavior Reproduction	Determines whether the model reproduces the behavior of the interest in the system	Model behaviors compared with behaviors of similar SD models	Performed well
8. Behavior Anomaly	Establishes the significance of important relationships by determining whether anomalous behavior arises when the relationship is deleted or modified	If assumptions are changed or communication overhead is removed the model exhibits anomalous behavior	Performed well
9. Family Member	The model can generate the behavior of other instances in the same class as the system the model was built to mimic	This model is built for a specific framework for agile software Development called as Scrum	N/A
10. Surprise Behavior	Finding unexpected behavior	Results imitate the expected behavior.	Performed well
11. Sensitivity Analysis	The impact of changing assumptions	Univariate and multivariate analysis were carried out by varying project size, number of personnel and productivity	Performed well

Name of the test	Purpose of the test	Procedures conducted in this research	Test Results
12. System Improvement	Modeling process helps change the system for the better	The focus of this study was not towards the improvement of the system	N/A

4.4. Results

A System dynamic model is a tool that allows recurrent exploration of the system, through changing assumptions and management policies [1]. This model was developed to investigate the factors affecting the Scrum framework when it is scaled for larger projects involving enormous workforce. Figure 21 shows the final iteration of the SD model developed for the Scrum framework. As an application of this model, it is simulated to analyze the difference between the basic Scrum framework and Scaled version of Scrum. This is simply done by varying number of personnel and the project size while keeping all other variables the same.

The main factors to affect Scrum performance as identified from SLR are the number of personnel, number of teams, communication overhead, training overhead and location of members i.e. whether project members are co-located or distributed. The model is simulated with the initial parameters as following:

Project size = 1000 tasks

Sprint velocity = 200 tasks / Sprint

Nominal Productivity = 0.4 tasks / day / person

The model is simulated with different staffing conditions and the resultant effect on project completion time and effort is noted. First the model is simulated with new and experienced personnel to be 10. The effect of this on work rate is shown in Figure 22. It can be noted that as the new staff is added to the project, the work rate (software development rate) decreases. However once the staff is trained and becomes experienced then the work rate increases until the project is completed. It can also be noted that communication overhead increases with an

increase in number of staff. The project completes in 199 days. Figure 23 shows the exhaustion of work backlogs to complete the project.

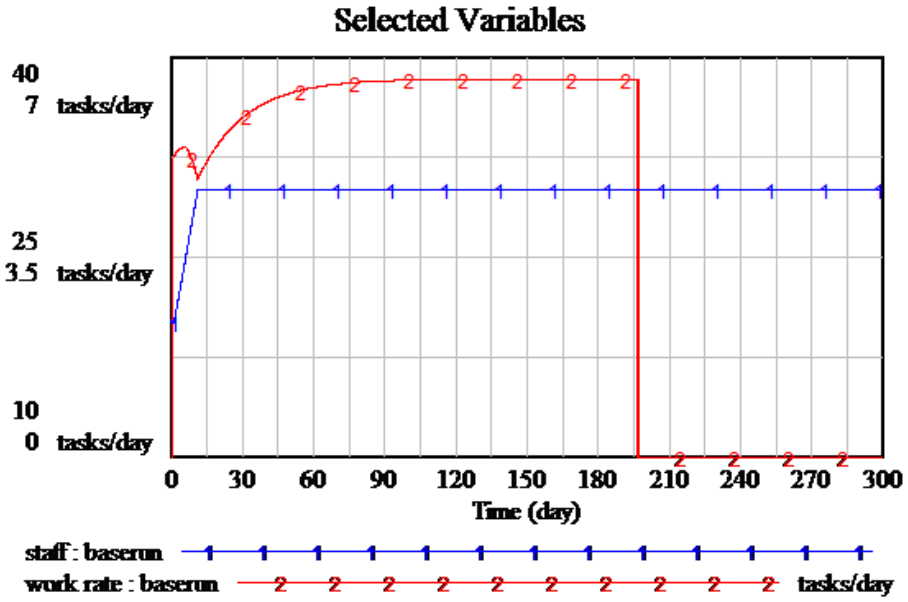

Figure 22 Graph plotted between number of Personnel and work rate

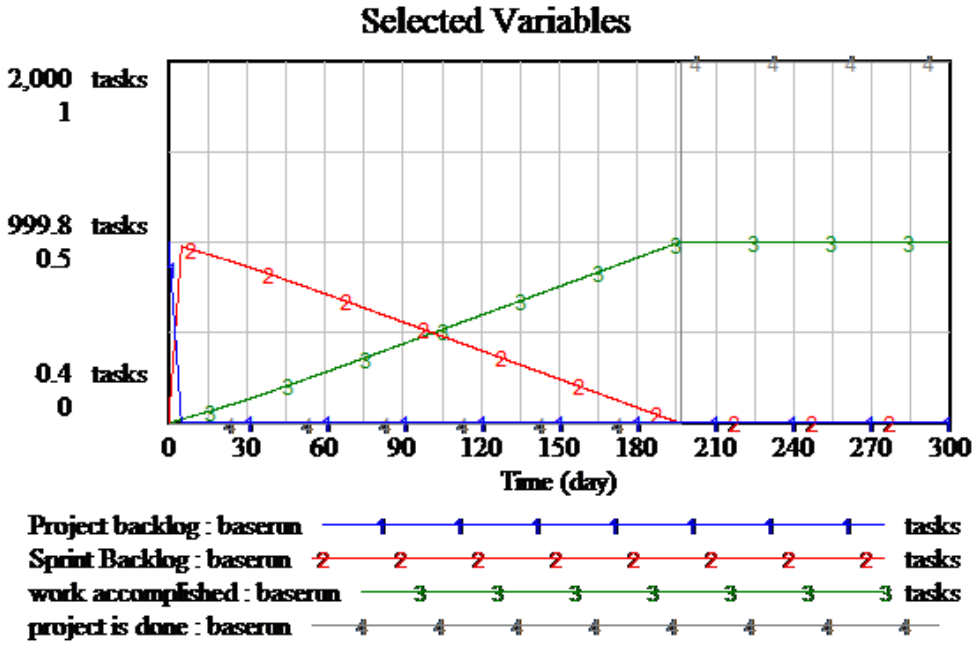

Figure 23 Work transfer in SD model

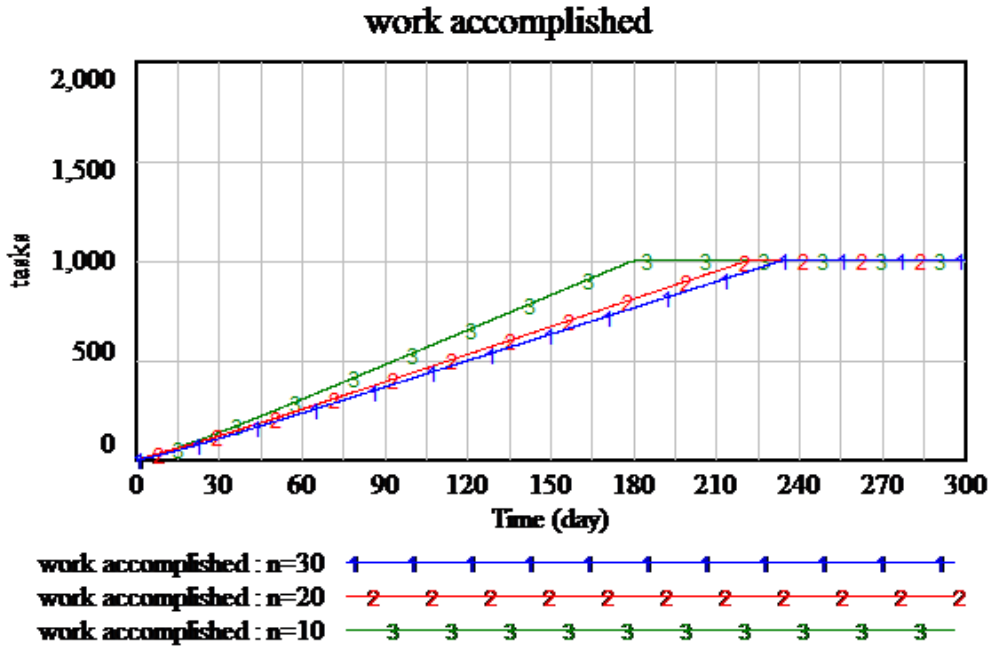

Figure 24 Project Completion time by increasing number of personnel (Project size=1000 tasks)

Figure 24 shows the decreasing project completion time with varying number of personnel. In this simulation, the project size is set to 1000 tasks. It can be noted that when the number of personnel are increased and all other variables are the same as previous run, the project completion time increases. This is due to the increase in communication overhead that is created with an increase in number of people working on the project. Moreover if the new personnel are added more and more, it results further decrease in work rate as more number of experienced personnel are training the new staff rather than working on the project.

For each of the simulation experiments, the results are monitored in the form of two project performance variables. The first performance variable is schedule, project completion time i.e. how long it will take to fully complete the project. The second one is the Cost, i.e. the amount of effort that will be required to complete the project. It is measured as the number of man-days required to complete the project. It can be seen in Figure 25 that as the number of personnel increases, the development effort also increases. The similar result is obtained when project size was increased to 5000 tasks as shown in Figure 26.

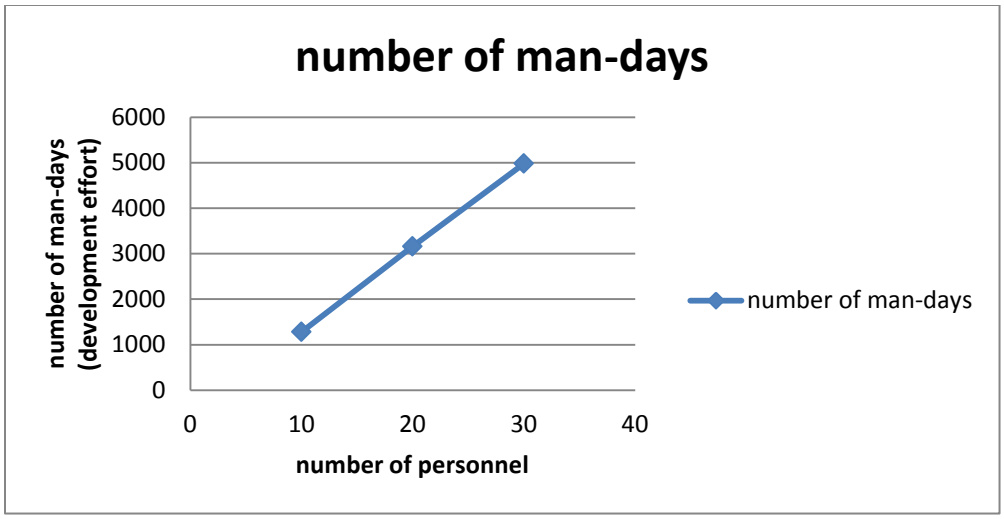

Figure 25 Development effort vs number of personnel

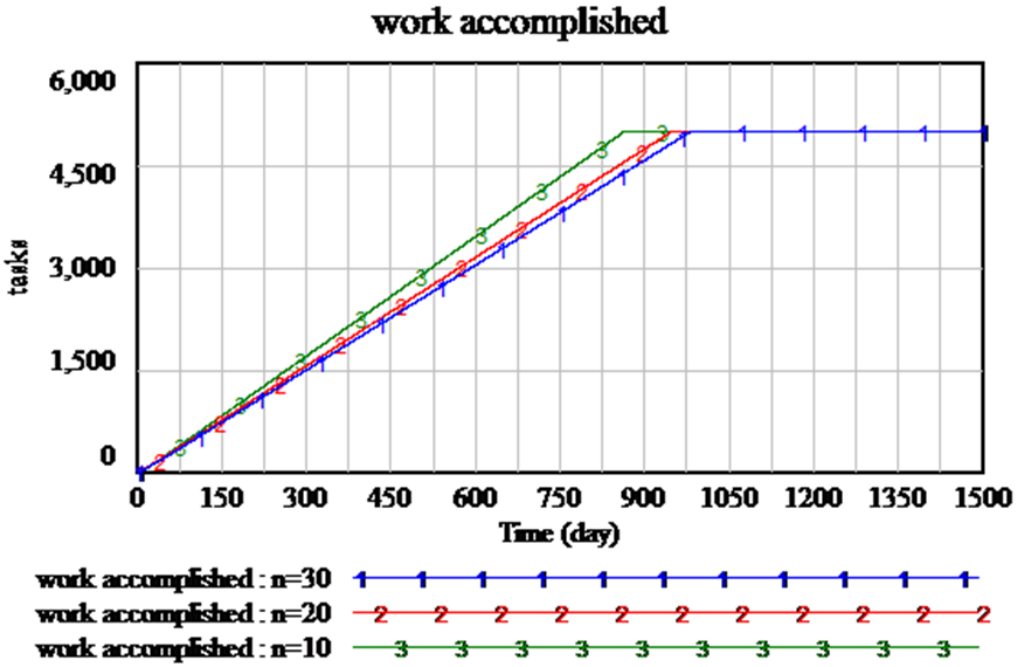

Figure 26 Project completion time by increasing number of personnel (project size=5000 tasks)

The proposed model and the analysis of the data suggest that increasing the number of personnel increases the productivity but at the same time communication overhead decreases the overall work rate. In order to keep the model simple, all the variables described in causal loop diagrams

were not included in simulation, so our study has been carried out under some limiting assumptions that could threaten its validity. The model is simple but can be considered as a genuine beginning point for further studies. The resultant behaviour of the simulation model observed is quite realistic, and its behaviour is the same also by changing the project size.

4.5. Discussion

This chapter presented a System dynamics model of the Scrum framework. The model is constructed in such a way so that it can be used for the basic Scrum framework as well as the scaled Scrum. This was simply done by increasing the project size and number of personnel. The scope of the model was limited to analyzing the impact of increasing the project personnel and observing the work rates. The difference in work rate for different values of personnel was due to the communication overhead and training overhead resulted from large teams. Results from the simulation process are in accordance to the information obtained from systematic literature review conducted. It also addresses the *RQ5* indicating that the main factor affecting the productivity in large teams is the communication. It is near impossible to remove the communication overhead in case of huge workforce but it can be reduced by using feature teams, face to face communication, high motivation and high team coordination towards a common goal rather than competition.

Chapter 5

Conclusions and Future Work

Scrum is the most popular agile framework used for project management used at the team level. Over the past decade, as its popularity grew, the industry began to scale Scrum to suit larger organizations and achieved mixed results. Many projects used Scrum successfully whereas few failed dramatically [47] and resulted in abandonment of the Scrum. In some vital and time critical projects, selection and adoption of software development process plays a very crucial role. In such projects, there is little or no room to undertake a risk that can cause project's failure. Therefore before adopting any development process, one must be confident of its results. This research has been carried out to provide insights and recommendations for how to scale Scrum to make it adaptable to large scale projects. These findings might help the developers to decide whether adopting Scrum for their projects is appropriate or not.

The objective of this thesis is to investigate management challenges in adopting scrum for large scale projects and to find and elaborate scaling practices used. A two-phased approach was adopted where in the first phase a systematic literature review was conducted and the second phase involved the development of a simulation model of the Scrum framework.

The presented SLR was successful in answering all the research questions shortlisted for this thesis. The practices identified through this SLR to scale Scrum for large projects are Scrum of Scrums, use of Area Product Owners, use of Proxy Product Owners, tailoring Scrum of Scrums, use of Feature backlog, use of Feature teams, meetings held between feature teams and Sprint Planning and review. Scrum of Scrums technique is widely used to scale the Scrum framework for large projects. The factors such as frequent SOS meetings, team collaboration, feature specific teams, feature backlog and ability to scale in recursive manner lead to the success of Scrum of Scrums technique. However the biggest challenge identified when using SOS was ensuring excellent communication between teams and team members. This might be difficult in distributed projects due to geographical, temporal distance and socio cultural differences but not impossible. In today's world where social networking is at boom and there are thousands of applications providing uninterrupted video conferencing at minimal rates, ensuring good

communication is no longer an impossible task. When teams are co-located or at least closely placed, even then if the team size and number of teams increase, it will be difficult to coordinate such large teams towards a common goal. The main challenge identified through this review was management of communication in large projects.

The presented SD simulation model reflected deeper understanding of the communication management in large scale projects. It was identified from SLR that a proper communication among teams is a major challenge while scaling Scrum. In second phase of thesis, System dynamics was used to analyze dynamic behavior of the Scrum framework by increasing number of personnel involved. First causal loop diagrams was devised to examine the relationships between various factors affecting the performance of the Scrum framework. These variables/factors were identified from the SLR and using judgemental estimation. The causal loop diagrams provided a clearer view of all the possible parameters which can alter the performance of the Scrum framework. In large scale Scrum, the main factors that lead to delay or drop in productivity are communication overhead, number of meetings, number of teams, diverse interest of team members, training overhead, and team collaboration. The relationships between all these factors are highlighted using a polarity symbol in causal loop diagrams. To keep the SD model simple, only a subset of these variables was selected to model in System dynamics.

The SD model developed successfully depicted the behavior of the Scrum framework by imitating the work flow between the project backlog, Sprint backlog and final product. The model was developed iteratively in small iterations to ensure the stability of the model at every step. The rework cycle was added to model to emulate the realistic behavior of the software projects. Due to the rework effects, a sample project that planned to complete in 125 days took up to 139 days. This project completion rate depends on the amount of work that is correct and complete which is modeled using the variable *fraction correct and complete (fcc).* Therefore by improving fcc and time to discover rework, the rework effect can be reduced. The SD model was culminated by adding the communication overhead and training overhead by increasing the number of members for different project sizes. The goal was to compare the effort and project completion time by varying number of personnel so that the influence of communication overhead can be analyzed. The model was successfully validated both in terms of its structure and behavior using practices recommended by Sterman [59]. In order to model the

communication overhead, Brooks's law [9] was incorporated in our model which has been discussed and analyzed extensively in the software engineering literature. The model gave predictable result that with an increase in number of members, there is a decrease in the work rate due to communication overhead. Moreover if the new employees are added late in the project, it affects more due to the training overhead required. However, smaller scale increase in staff will help to reduce the explosive increase in communication overhead. It was observed from systematic literature review and causal loop diagrams, number of meetings also play an important role in the SOS technique for large scale Scrum as when extensively performed can cause a delay in the project completion time. This variable was not modeled in the SD model due to lack of realistic dataset available from real time software projects. Therefore this study has been carried out under some limiting assumptions that could threaten its validity. The model developed can be considered as a genuine beginning point for further studies. The resultant behaviour of the simulation model observed is quite realistic and showed a similar behavior even when project size was changed. The SD model developed support the results obtained from SLR that even though the Scrum framework can be used for developing large projects but can cause communication problems for very large teams. The findings of this research can direct the developers to focus on the right issues from the beginning of the project which in turn can reduce development effort and cost.

Finally this chapter concludes by providing some ideas to extend this research in future. The model developed in this thesis represents a nominal case, and would require calibration to specific project environments. The proposed model needs to be further elaborated and validated. The model can be validated against data obtained from any real time software development project developed by some company or enterprise. This can give deeper insights and might also improve the model. The model can be extended by including all the variables described in causal loop diagrams. System dynamic modeling and simulation is an inexpensive way to gain deep insights when real time data is unavailable. In this thesis, only the Scrum framework was considered to develop large projects. Similar kind of studies can be carried out for other agile methodologies.

Appendix

Studies selected for Systematic Literature Review

[S1] Paasivaara, M., & Lassenius, C. (2011, September). Scaling scrum in a large distributed project. In *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on* (pp. 363-367). IEEE.

[S2] Hossain, E., Bannerman, P., & Jeffery, R. (2011). Towards an understanding of tailoring scrum in global software development: a multi-case study. *International Conference on Software and Systems Process*, 110–119.

[S3] Lagerberg, L., Skude, T., Emanuelsson, P., Sandahl, K., & Stahl, D. (2013). The Impact of Agile Principles and Practices on Large-Scale Software Development Projects: A Multiple-Case Study of Two Projects at Ericsson. 2013 *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, 348–356.

[S4] Paasivaara, M., Lassenius, C., & Heikkila, V. T. (2012, September). Inter-team coordination in large-scale globally distributed scrum: Do Scrum-of-Scrums really work? , In *Empirical Software Engineering and Measurement (ESEM), 2012 ACM-IEEE International Symposium on* (pp. 235-238). IEEE.

[S5] Maranzato, R. P., Neubert, M., & Herculano, P. (2011). Moving back to scrum and scaling to scrum of scrums in less than one year. *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion - SPLASH '11*, New York, NY, USA, 125-130. doi:10.1145/2048147.2048186

[S6] Hannay, J. E., & Benestad, H. C. (2010). Perceived productivity threats in large agile development projects. *In Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '10). ACM, New York, NY, USA, Article 15, 10 pages. http://doi.acm.org/10.1145/1852786.1852806.*

[S7] Paasivaara, M., Heikkila, V. T., & Lassenius, C. (2012). Experiences in Scaling the Product Owner Role in Large-Scale Globally Distributed Scrum. *2012 IEEE Seventh International Conference on Global Software Engineering*, 174–178. doi:10.1109/ICGSE.2012.41

[S8] Lyon, R., & Evans, M. (2008). Scaling Up Pushing Scrum out of its Comfort Zone. *Agile 2008 Conference*, 395–400.

[S9] Babinet, E., & Ramanathan, R. (2008, August). Dependency management in a large agile environment. In *Agile 2008 Conference* (pp. 401-406). IEEE.

[S10] Paasivaara, M., Durasiewicz, S., & Lassenius, C. (2008). Distributed Agile Development: Using Scrum in a Large Project. *2008 IEEE International Conference on Global Software Engineering*, 87–95. doi:10.1109/ICGSE.2008.38

[S11] Badampudi, D., Fricker, S. A., & Moreno, A. M. (2013). Perspectives on Productivity and Delays in Large-Scale Agile Projects, *Agile Processes in Software Engineering and Extreme Programming*, Volume 149, pp 180-194. Springer Berlin Heidelberg

References

[1] Abdel-Hamid, T., & Madnick, S. E. (1991). *Software project dynamics: an integrated approach*. Prentice-Hall, Inc.

[2] Abrahamsson, P., Salo, O., Ronkainan, J., & Warsta, J. (2002). *Agile Software Development Methods: Review and Analysis.* VTT Technical report, p. 107 .

[3] Abrahamsson, P. (2007, October). AGILE Software Development of Embedded Systems. In *ITEA2 symposium, Berlin, Germany* (pp. 18-19).

[4] Babinet, E., & Ramanathan, R. (2008, August). Dependency management in a large agile environment. In *Agile 2008 Conference* (pp. 401-406). IEEE.

[5] Badampudi, D., Fricker, S. A., & Moreno, A. M. (2013). Perspectives on Productivity and Delays in Large-Scale Agile Projects, *Agile Processes in Software Engineering and Extreme Programming,* Volume 149, pp 180-194. Springer Berlin Heidelberg.

[6] Beck K., Beedle M., Bennekum A., Cockburn A., Cunningham W., Fowler M., Grenning J., Highsmith J., Hunt A., Jeffries R., Kern J., Marick B., Martin R., Mellor S., Schwaber K., Sutherland J., Thomas D. The Agile Manifesto: 2001, http://agilemanifesto.org/ last visited on March 12, 2015.

[7] Begel, A., & Nagappan, N. (2007, September). Usage and perceptions of agile software development in an industrial context: An exploratory study. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on* (pp. 255-264). IEEE.

[8] Boehm, B. W., Madachy, R., & Steece, B. (2000). *Software cost estimation with Cocomo II with Cdrom*. Prentice Hall PTR.

[9] Brooks, F. P. (1975). *The mythical man-month* (Vol. 1995). Reading, MA: Addison-Wesley.

[10] Cao, L., Ramesh, B., & Abdel-Hamid, T. (2010). Modeling dynamics in agile software development. *ACM Transactions on Management Information Systems (TMIS), 1, 1, Article 5*.

[11] Cardozo, E., Neto, J. B. F. A., Barza, A., França, A., & da Silva, F. (2010, April). SCRUM and productivity in software projects: a systematic literature review. In *14th International Conference on Evaluation and Assessment in Software Engineering (EASE)*.Cliffs, NJ

[12] Cloke G. (2007), "Get Your Agile Freak On! Agile Adoption at Yahoo! Music", in *AGILE 2007*, 2007, pp. 240-248.

[13] Cocco, L., Mannaro, K., Concas, G., & Marchesi, M. (2011). Simulating kanban and scrum vs. waterfall with system dynamics. In *Agile Processes in Software Engineering and Extreme Programming* (pp. 117-131). Springer Berlin Heidelberg.

[14] Cohn, M. (2005). *Agile estimating and planning*. Pearson Education.

[15] Cruzes, D. S., & Dybå, T. (2011, September). Recommended steps for thematic synthesis in software engineering. In *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on* (pp. 275-284). IEEE.

[16] Deemer,P., Benefield, G., Larman, C., Vodde, B.(2008). The Scrum Primer Version 2.0, Scrum Training Institute, available at http://www.scrumprimer.com ,last visited on August 19, 2015.

[17] Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and software technology*, *50*(9), 833-859.

[18] Eskelinen Arto et al. (2010), Scrum Master Course, lecture notes by Reaktor Innovations Oy, Course at Ericsson, Kirkkonummi Finland. http://reaktor.com/training/ Last visited on January 12, 2015.

[19] Forrester, J. W. (1995). The beginning of system dynamics. *McKinsey Quarterly*, 4-17.

[20] Forrester, J. W., & Senge, P. M. (1996). Tests for building confidence in system dynamics models. *Modelling for management: simulation in support of systems thinking*, *2*, 414-434.

[21] Glaiel, F., Moulton, A., & Madnick, S. (2013). Agile project dynamics: A system dynamics investigation of agile software development methods. In *31st International Conference of the System Dynamics Society*, pp. 53-63.

[22] Greene, B. (2004, June). Agile methods applied to embedded firmware development. In *Agile Development Conference, 2004* (pp. 71-77). IEEE.

[23] Hannay, J. E., & Benestad, H. C. (2010). Perceived productivity threats in large agile development projects. *In Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '10). ACM, New York, NY, USA, Article 15, 10 pages. http://doi.acm.org/10.1145/1852786.1852806.*

[24] Heikkilä, V. T., Paasivaara, M., Lassenius, C., & Engblom, C. (2013). *Continuous release planning in a large-scale scrum development organization at Ericsson* (pp. 195-209). Springer Berlin Heidelberg.

[25] Herbsleb, J. D., & Moitra, D. (2001). Global software development. *Software, IEEE*, *18*(2), 16-20.

[26] Highsmith, J. A. (2002). *Agile software development ecosystems* (Vol. 13). Addison-Wesley Professional.

[27] Hossain, E., Babar, M. a., & Paik, H. P. H. (2009). Using Scrum in Global Software Development: A Systematic Literature Review. *2009 Fourth IEEE International Conference on Global Software Engineering*, 175–184. doi:10.1109/ICGSE.2009.25

[28] Hossain, E., Bannerman, P., & Jeffery, R. (2011). Towards an understanding of tailoring scrum in global software development: a multi-case study. *International Conference on Software and Systems Process*, 110–119.

[29] http://www.scrum-institute.org/The_Scrum_Product_Backlog.php last visited February 10, 2015.

[30] Kellner, M. I., Madachy, R. J., & Raffo, D. M. (1999). Software process simulation modeling: why? what? how?. *Journal of Systems and Software*, *46*(2), 91-105.

[31] Kitchenham, B. (2004). Procedures for performing systematic reviews. *Keele, UK, Keele University*, *33*(2004), 1-26. http://people.ucalgary.ca/~medlibr/kitchenham_2004.pdf. Last visited on February 25, 2015

[32] Kniberg, H. (2007). *Scrum and XP from the Trenches: How we do Scrum*. http://www.lulu.com/content/899349. Last visited on October 9, 2014.

[33] Laanti, M., Salo, O., & Abrahamsson, P. (2011). Agile methods rapidly replacing traditional methods at Nokia: A survey of opinions on agile transformation. *Information and Software Technology*, *53*(3), 276-290.

[34] Lagerberg, L., Skude, T., Emanuelsson, P., Sandahl, K., & Stahl, D. (2013). The Impact of Agile Principles and Practices on Large-Scale Software Development Projects: A Multiple-Case Study of Two Projects at Ericsson. 2013 *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, 348–356.

[35] Larman, C., & Vodde, B. (2008). *Scaling lean & agile development: thinking and organizational tools for large-scale Scrum*. Pearson Education.

[36] Larman, C., and Vodde, B. (2010). *Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum* (1st ed.). Addison-Wesley Professional.

[37] Lyon, R., & Evans, M. (2008). Scaling Up Pushing Scrum out of its Comfort Zone. *Agile 2008 Conference*, 395–400.

[38] Madachy, R. J. (2007). *Software process dynamics*. John Wiley & Sons.

[39] Maranzato, R. P., Neubert, M., & Herculano, P. (2011). Moving back to scrum and scaling to scrum of scrums in less than one year. *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion - SPLASH '11*, New York, NY, USA, 125-130. doi:10.1145/2048147.2048186

[40] Martin, R. H., & Raffo, D. (2000). A model of the software development process using both continuous and discrete models. *Software Process: Improvement and Practice, 5*(2-3), 147-157.

[41] Oliva, R. (2003). Model calibration as a testing strategy for system dynamics models. *European Journal of Operational Research, 151*(3), 552-568.

[42] Paasivaara, M., & Lassenius, C. (2011, September). Scaling scrum in a large distributed project. In *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on* (pp. 363-367). IEEE.

[43] Paasivaara, M., Durasiewicz, S., & Lassenius, C. (2008). Distributed Agile Development: Using Scrum in a Large Project. *2008 IEEE International Conference on Global Software Engineering*, 87–95. doi:10.1109/ICGSE.2008.38

[44] Paasivaara, M., Durasiewicz, S., & Lassenius, C. (2009, July). Using scrum in distributed agile development: A multiple case study. In *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on* (pp. 195-204). IEEE.

[45] Paasivaara, M., Heikkila, V. T., & Lassenius, C. (2012). Experiences in Scaling the Product Owner Role in Large-Scale Globally Distributed Scrum. *2012 IEEE Seventh International Conference on Global Software Engineering*, 174–178. doi:10.1109/ICGSE.2012.41

[46] Paasivaara, M., Lassenius, C., & Heikkila, V. T. (2012, September). Inter-team coordination in large-scale globally distributed scrum: Do Scrum-of-Scrums really work? , In *Empirical Software Engineering and Measurement (ESEM), 2012 ACM-IEEE International Symposium on* (pp. 235-238). IEEE.

[47] Ralph, P., & Shportun, P. (2013, June). Scrum Abandonment in Distributed Teams: A Revelatory Case. In *PACIS* (p. 42).

[48] Randers, J. (Ed.). (1980). *Elements of the system dynamics method* (pp. 117-139). Cambridge, MA: MIT press.

[49] Reifer, D.J.; Maurer, F.; Erdogmus, H.(2003), "Scaling agile methods," *Software, IEEE* , vol.20, no.4, pp.12,14, July-Aug. 2003

[50] Schwaber K.(2008), It's Not Scrum If..., Presentation at Stockholm Scrum Gathering Fall 2008, available at http://www.scrumalliance.org/resource_download/441. Last accessed on March 18 2014.

[51] Schwaber K., 2009, Scrum Guide, Online Guide, available at http://www.scrumalliance.org/resource_download/598. Last accessed on August 5, 2015

[52] Schwaber K., Beedle M. (2001). Agile software development with scrum. Prentice Hall PTR, Upper Saddle River, NJ, USA

[53] Schwaber K., 2014, Ken Shwaber's Blog: Telling it like it is. Do you know how to Scale Scrum? https://kenschwaber.wordpress.com/2014/11/13/do-you-know-how-to-scale-scrum/ Last accessed on April 10, 2015

[54] Schwaber, K. (2004). *Agile project management with Scrum*. Microsoft Press.

[55] Schwaber, K. (2007). *The enterprise and scrum*. Microsoft Press.

[56] Schwaber, K., & Beedle, M. (2002).Agile Software Development with Scrum. Prentice Hall

[57] Scrum in Larger Organizations, Part 1 An Interview with Jeff Sutherland by SoftHouse. Retrieved from http://eng.softhouse.se/downloads/Intervju_Pdf_Small.pdf. Last accessed on July 28, 2015

[58] Seikola, M. (2010). *The Scrum Product Backlog as a Tool for Steering the Product Development in a Large-Scale Organization* (Doctoral dissertation, Aalto University)

[59] Sterman, J. D. (2000). *Business dynamics: systems thinking and modeling for a complex world* (Vol. 19). Boston: Irwin/McGraw-Hill.

[60] Sutherland, J. (2001). Agile can scale: Inventing and reinventing scrum in five companies. *Cutter IT journal*, *14*(12), 5-11.

[61] Sutherland, J., Schwaber, K., & Sutherland, C. J. (2007). The scrum papers: Nuts, bolts, and origins of an Agile Process. Boston: Scrum, Inc.

[62] Vensim PLE (version 6.3) downloaded from http://vensim.com/free-download/. Last visited on March, 2014.

[63] Walker, J., (2013), How to Apply Agile Techniques to Distributed Teams and Large Projects. Beyond Scrum. http://www.perforce.com/company/newsletter/2013/01/beyond-scrum-how-apply-agile-techniques-distributed-teams-large-projects. Last accessed on March 25, 2015

[64] West, D. (2011). Water-scrum-fall is the reality of agile for most organizations today. *Forrester Research*. http://www.storycology.com/uploads/1/1/4/9/11495720/water-scrum-fall.pdf. Last accessed on November 10, 2014

[65] www.mountaingoatsoftware.com/agile/scrum/sprint-backlog.Last accessed on January 12, 2015.