Lappeenranta University of Technology
School of Industrial Engineering and Management
Software Engineering and Information Management
Department of Master Degree Program in Computer Science

Mehar Ullah

Comparison and problems between Traditional and Agile software development methods

Supervisors: Professor Kari Smolander

Associate Professor Erja Mustonen-Olli

ABSTRACT

Lappeenranta University of Technology
School of Industrial Engineering and Management
Software Engineering and Information Management
Department of Master Degree Program in Computer Science

Mehar Ullah

Comparison and problems between Traditional and Agile software development methods

Master’s Thesis, November 13, 2014

76 Pages, 14 Figures, 7 Tables

Supervisors: Professor Kari Smolander

Associate Professor Erja Mustonen-Ollila

Key Words: Agile Methods, Software Engineering, Traditional Development

In today’s world because of the rapid advancement in the field of technology and business, the requirements are not clear, and they are changing continuously in the development process. Due to those changes in the requirements the software development becomes very difficult. Use of traditional software development methods such as waterfall method is not a good option, as the traditional software development methods are not flexible to requirements and the software can be late and over budget. For developing high quality software that satisfies the customer, the organizations can use software development methods, such as agile methods which are flexible to change requirements at any stage in the development process. The agile methods are iterative and incremental methods that can accelerate the delivery of the initial business values through the continuous planning and feedback, and there is close communication between the customer and developers. The main purpose of the current thesis is to find out the problems in traditional software development and to show how agile methods reduced those problems in software development. The study also focuses the different success factors of agile methods, the success rate of agile projects and comparison between traditional and agile software development.

ACKNOWLEDGEMENTS

I would like to convey my thanks message to all who guide, support, encourage and advised me throughout my stay here in Finland during my course work and also in my thesis time.

First of all I would like to convey my thanks to supervisors Professor Kari Smolander and Associate Professor Erja Mustonen-Ollila. Their advices and support make me able to complete my thesis task. During my first semester when I was very confused Associate Professor Erja Mustonen-Ollila encouraged me a lot. I would also like to thank Department of Software Engineering and Information Management for giving such a nice study environment and support throughout my studies.

Moreover, I would like to say thanks to my Mom and Dad, my elder brother Adnan Ahmad and my wife Kiran and son Hamdan for supporting me throughout my studies.

I am also so much thankful to Suvi Tiainen, all the staff of Lappeenranta University of Technology for providing me all the facilities and services during my studies.

Lappeenranta, November 13, 2014

Mehar Ullah

TABLE OF CONTENTS

Contents

LIST OF ABBREVATIONS 6
LIST OF FIGURES 7
LIST OF TABLES 8
1. Introduction 9
1.1. Background 9
1.2. Goals of the thesis 10
1.3. Research questions 11
1.4. Scope and limitations 12
1.5. Structure and Research Method of the Thesis 12
1.5.1. Structure of the Thesis 12
1.5.2. Research Method 14
2. An overview of Traditional Software Development Process Models 15
2.1. Traditional Software development process models 15
2.2. The waterfall process model 18
2.2.1. Requirement definition 19
2.2.2. System and software design 20
2.2.3. Implementation and unit testing 20
2.2.4. Integration and system testing 20
2.2.5. Operation and maintenance 20
2.3. Advantages and Disadvantages of waterfall model 20
3. Different types of agile methods 22
3.1. Agile Software Development Alliance and Agile Manifestive 23
3.2. Agile methods 24
3.3. Extreme Programming (XP) 26
3.3.1. XP process phases 27
3.3.2. Roles and responsibilities in XP 30
3.3.3. XP Practices 31

3.3.4. XP usability ... 33
3.4. Scrum ... 33
3.4.1. SCRUM Phases ... 35
3.4.2. The Scrum Team's Roles and Responsibilities ... 36
3.4.3. Scrum usability ... 38
3.4.4. Scrum Terminology ... 38
3.5. Rational Unified Process (RUP) ... 42
3.5.1. The Rup Process Phases ... 43
3.5.2. RUP Workflow Dimensions ... 45
3.5.3. Best Practices of RUP ... 47
3.5.4. The roles and responsibilities in RUP ... 48
3.5.5. RUP usability ... 50
4. Agile methods in Software Development ... 51
4.1. Difference between Traditional and Agile software development ... 51
4.2. Project success or failure rate of Traditional and Agile projects ... 54
4.3. Success Factors in Agile Projects ... 57
4.3.1. Requirements and Customer involvement ... 60
4.3.2. Close communication ... 60
4.3.3. Continuous integration and Early Testing ... 62
4.3.4. Small release ... 63
5. Discussion and conclusions ... 64
5.1. Discussion ... 64
5.2. Conclusions ... 66
5.3. Future Studies ... 67
References ... 69

LIST OF ABBREVATIONS

DK	Development Kit
GUI	Graphical User Interface
RQ	Research Questions
RUP	Rational Unified Process
SCM	Software Configuration Management
SDLC	System Development Life Cycle
SRS	Software Requirement Specifications
SQA	Software Quality Assurance
UML	Unified Modeling Language
V & V	Verification and Validation
XP	Extreme Programming
UC	Use Case
SA	Software Architecture
LD	Logical Diagram
CS	Conceptual Schema
E-Business	Electronic Business
E-Commerce	Electronic Commerce

LIST OF FIGURES

Figure 1: Structure of the Thesis

Figure 2: Software process

Figure 3: Software life cycle

Figure 4: The Waterfall model steps

Figure 5: XP process

Figure 6: Iteration in XP development process

Figure 7: The Scrum Process

Figure 8: RUP phases and workflows

Figure 9: Comparison between Roles in RUP

Figure 10: Requirement categories

Figure 11: Agile project success rate

Figure 12: Hypothesized success factors

Figure 13: Agile communication cycle

Figure 14: The communication value of different techniques

LIST OF TABLES

Table 1: XP Practices

Table 2: XP Practices by 5-A Model

Table 3: Terminologies used in SCRUM communication

Table 4: Difference between agile and traditional software development

Table 5: Success and failure rate of Agile and Traditional projects

Table 6: Comparison rate of Agile and Traditional projects

Table 7: Success and Failure of Global Projects

1. Introduction

1.1. Background

Nowadays in any field like business, education, sports etc. the success depend on the software being used, due to the fast development in technology the organizations needs its software updated and the software should meet all the business needs. The rapid growing completions between the organizations have created a challenge for the software development companies (Czarnacka-Chrobot, 2010).

The organizations need its business software on time with in the budget and also during the development time the organizational requirements changes due to the change in the business process, which creates problems for the development team as it become very difficult and expensive for the development team to make changes in the middle of the development phase and thus the software become over budget and late (Awad, 2005).

In order to overcome those problems in the software development the software development companies must have to use flexible software development methods that can accommodate any changes at any stage of the software development. Agile development is very flexible for changed requirements at any stage of the software development and that’s why very suitable for the organizations to take the benefit from the agile methods (Devi, 2013).

The agile software process is the combination of the best practices and their previous success and failure experiences with many software development projects regarding what works and what does not. Each of these two practitioners (best practice and previous success and failure experience) had their own different philosophies about how they approached software development. However, all of them advocated close collaboration between software development and business teams, as opposed to silo development by software teams; face-to-face communication, as opposed to over-emphasis on written documentation in projects; frequent delivery of portions of working software, as opposed to final delivery of the complete product at the end; accepting changing requirements by customers, as opposed to defining a fixed set of requirements “cast-in-stone”; and adaptive organizational capability of teams according to changing business requirements (Misra et al., 2009)

The word agile means light weight; the main theme of agile method is the simplicity and speed. The main points of agile methods according to Fowler and Highsmith (2001) are *Incremental* (small software releases, with rapid cycles) *Cooperative* (customer and developers working constantly together with close communication) *Straightforward* (the method itself is easy to learn and to modify, well documented) *Adaptive* (able to make last moment changes).

The usage of the agile methods has increased from the agile manifesto (Fowler and Highsmith, 2001) in March 2001, when 17 members from different software development methodologies gathered and developed the agile software development alliance.
A few main points of this manifesto are as follows.

- Satisfying the customer by providing the working software on time.
- Welcome changes in the requirements during the software development either in the early stages or even late in the development.
- The software development should be quick enough to satisfy the customer requirements and the development should either be in weeks or in months.
- The interaction of the business people with the developers of the project.

1.2. Goals of the thesis

The goals of the thesis are to point out the problems in the traditional software development methods like waterfall etc. Due to which the software become over budget and late. The thesis also focuses on few agile methods and the success factors of agile methods. How agile methods reduce the problems in software development. The comparison between traditional and agile methods has also been discussed in the thesis.

The thesis has five chapters. The *First chapter* of the thesis is about the introduction that contains the background knowledge, goals of the thesis, research questions, scope and limitations of the thesis, and structure of the thesis. The *second* chapter of the thesis shows an overview of the traditional software development methods and problems in the traditional software development method using the waterfall process model. Advantages and disadvantages of waterfall method are presented in this chapter. In the second chapter the first research question of the thesis is explained. In the *third* chapter of the thesis three agile methods has been explained, in the same chapter three, the success factors of agile methods are discussed. The second research question of the thesis is explained in the third chapter. The *fourth* chapter of the thesis focuses the difference between traditional software development and agile software development and how the agile methods are used to reduce the cost and time in the software development. The comparison of success and failure rate of agile and traditional projects is discussed in fourth chapter. The fourth chapter explains the third and fourth research questions of the thesis. In Chapter *five* the discussion and conclusion are presented and the future work is also discussed in this chapter.

1.3. Research questions

Below are the four research questions (RQ) that can cover the above goals of the thesis.

RQ 1: What are the problems in traditional software development methods (like waterfall)?

RQ 2: What are the main agile methods in software development?

RQ 3: How to compare the agile methods with traditional methods (like waterfall) in software development?

RQ 4: How agile methods reduce problems in software development?

1.4. Scope and limitations

“Software engineering or engineering discipline is concerned with all aspects of software production from the early stage of system specification to maintaining the system after it has gone into use” (Sommerville, 2007). Software engineering is producing the software for organizations to full fill their requirements. As clear from the definition software engineering is a large field that can cover many processes and phases.

There are many methods for software development. Some of the methods are considered good as compared to another method and vice versa. The current thesis is focused on the weaknesses of traditional software development specially the budget and time. The thesis also focuses on the new agile methods for the software engineering and how the agile methods overcome the problems that were there in traditional software engineering methods.

The current thesis is limited to the traditional waterfall modal of software engineering and the agile methods as there were limited articles about the comparison of waterfall method and agile methods.

1.5. Structure and Research Method of the Thesis

1.5.1. Structure of the Thesis

The following Figure 1 shows the structure of the thesis.

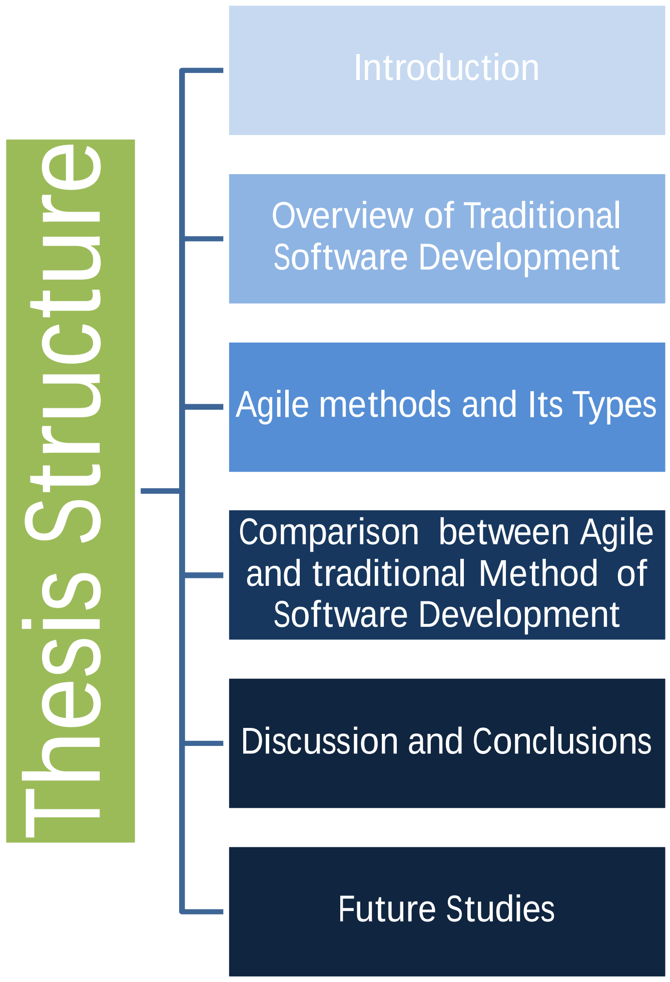
Thesis Structure
Introduction
Overview of Traditional Software Development
Agile methods and Its Types
Comparison between Agile and traditional Method of Software Development
Discussion and Conclusions
Future Studies

Figure 1: Structure of the Thesis.

First section of the thesis is introducing the topic, background knowledge about the software development using the traditional methods like waterfall method and later on the agile methods have been explained and how agile were introduced and why. In this section the objective of the thesis, the research questions, scope and limitation of the thesis and the structure and research method of the thesis are mentioned.

The *second section* of the thesis is about the tradition development using the waterfall development model. The waterfall life cycles are explained and later on at the end of this potion the advantages and disadvantages of waterfall method is described.

The *third section* of the thesis discusses the need of the agile methods, how the agile software development alliance is made, how the agile manifestive have been introduced, the points of agile manifestive are described, the purpose of the agile manifestive is discussed, the agile method is defined, three main agile methods Extreme Programming (XP). SCRUM and Rational Unified Process (RUP) are explained; the phases, roles and responsibilities, when the process can be used are mentioned in details.

The *fourth section* of the thesis the difference between the agile and traditional software development model are outlined, the project success and failure rate are mentioned here, the global projects success and failure were described, some of the main factors in agile methods that can reduced the time and cost in agile projects are explained.

The *fifth section* of the thesis is about the discussion and conclusions. In this section the complete summary of the thesis is presented.

The *sixth* section is about the future studies.

1.5.2. Research Method

The research method used in this thesis is qualitative research approach. A qualitative research is one that is not experimentally examined or measured (Denzin and Lincoln, 2011).In this thesis the qualitative research approach is used by using the literature surveys, research articles published in different journals related to the information technology and specially software engineering. Different books related to software engineering, software cost estimation have been used and referenced. The main database used in the thesis is IEEE and SCOPUS. Google have been used for a few topics. Collected literature is based on the key words like traditional software development, waterfall method, agile methods and history of agile methods, success factors of agile methods, reducing time and cost in agile methods, continuous integration, small releases etc.

2. An overview of Traditional Software Development Process Models

In this chapter the first research question of the thesis is explained.

2.1. Traditional Software development process models

Software development Process models are used for all most all type of software development projects and according to Sommerville (2011) a software model is the simplified form of a software process that represents a particular perspective and provides partial information about the process.

Software process is a framework of activities that are involved in all most all the software projects regardless of the size and complexity of the tasks (Pressman, 2001).See Figure 2.

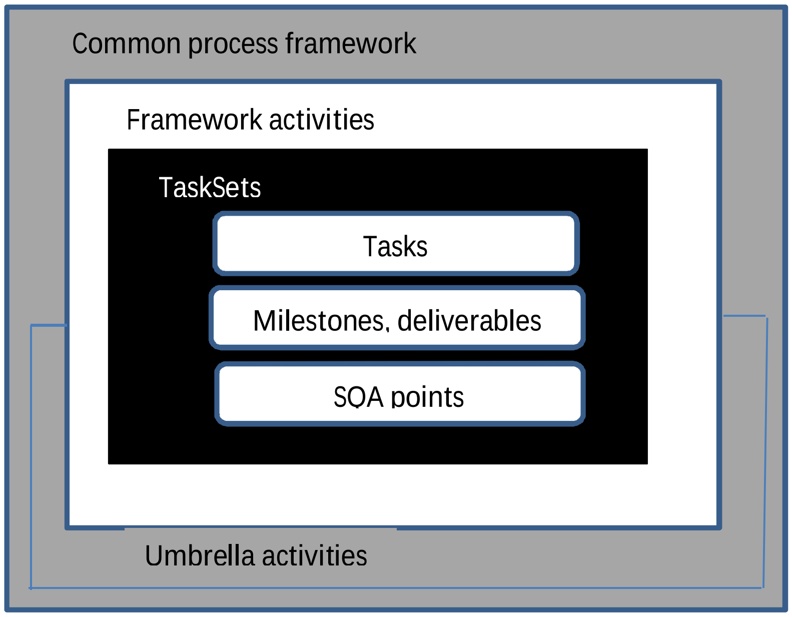

Figure 2: Software process (Pressman, 2001).

Figure 2 explains a software process, and we can see that software process consists of set of activities that contains the tasks, the output of those tasks is called milestones and deliverables and contains the software quality assurance (SQA) points. Software is called high quality software if it meets the required requirements. The Umbrella activities consist of SQA, software configuration management (SCM) and measurement. The umbrella activities are independent of all the frame work activities.

Currently there are many software processes used for software development, but according to Sommerville (2011) there are some phases that are almost the same in every process. Those phases are listed below.

- **Software specifications**: Here the functionality of the software is defined and the constraints on the operations of those functionalities are defined.
- **Software design and implementation**: The software which can complete the required specification can be produced.
- **Software validation**: The software should satisfy the customer, means the software should perform all the tasks for which it is produced.
- **Software evolution**: The software must be ready to meet the changing customer need.

According to Maciaszek and Liong (2005) a software product in its development life cycle is either in phasing-in stage or in phasing-out stage. Phasing-in stage is the initial startup stage of the product during which the requirements for the software are gathered. Based on those requirements the rest of the software development phases (system design, implementation, and integration and deployment) starts. Phasing-out is the last stage of the software product. Phasing-out stage begins at the operation and maintenance phase as shown in Figure 3a.

Once a company introduces software product in their organization then that software exist for ever in the organization with some changes and the organization is not able to run its work manually. With the passage of time there occurs many changes in the same software product and the software changes to a legacy software, and either some components or the whole system is in the phasing-out stage and the thus a new life cycle starts.

As shown in Figure 3b the phasing-out of the old system becomes the phasing-in of the new system and a new life cycle of the system starts. The old system still works till the new system start working.

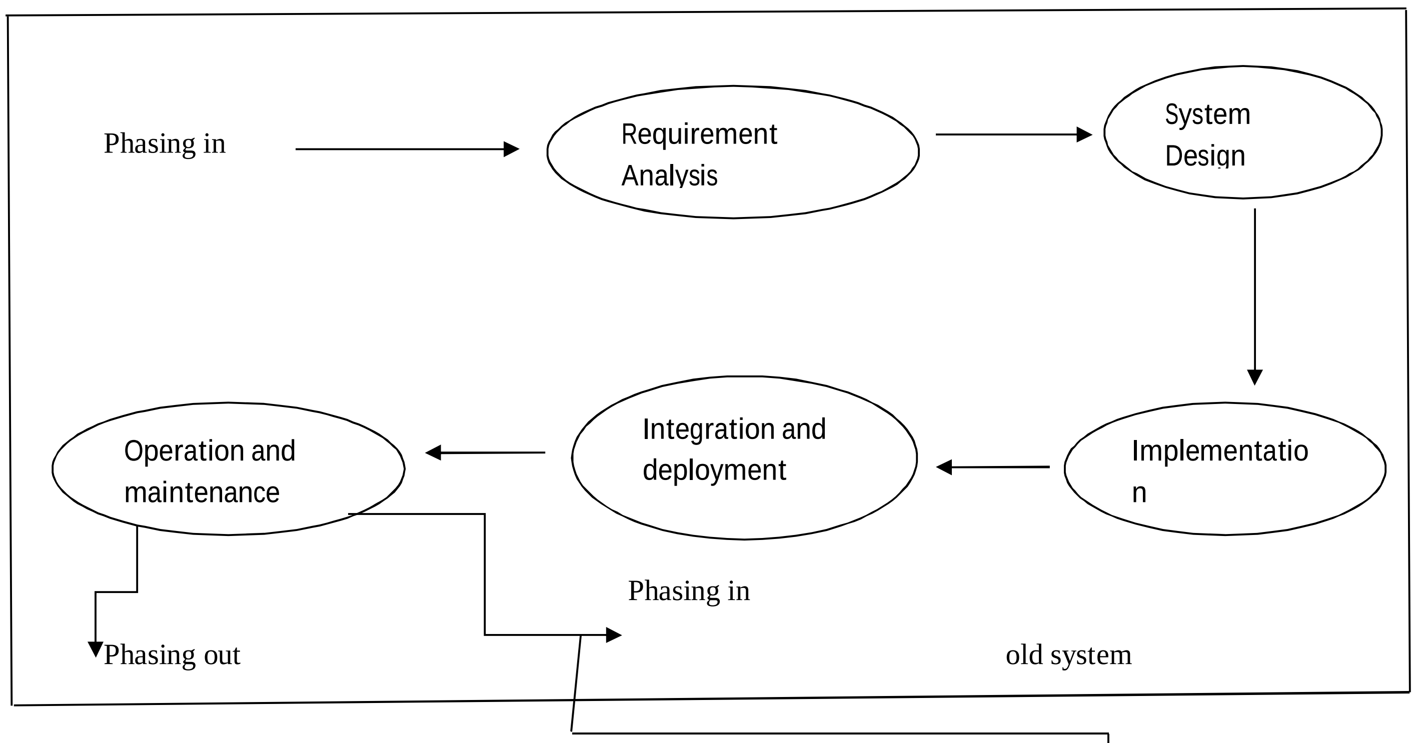

Figure 3a: Software development life cycle (Maciaszek and Liong, 2005)

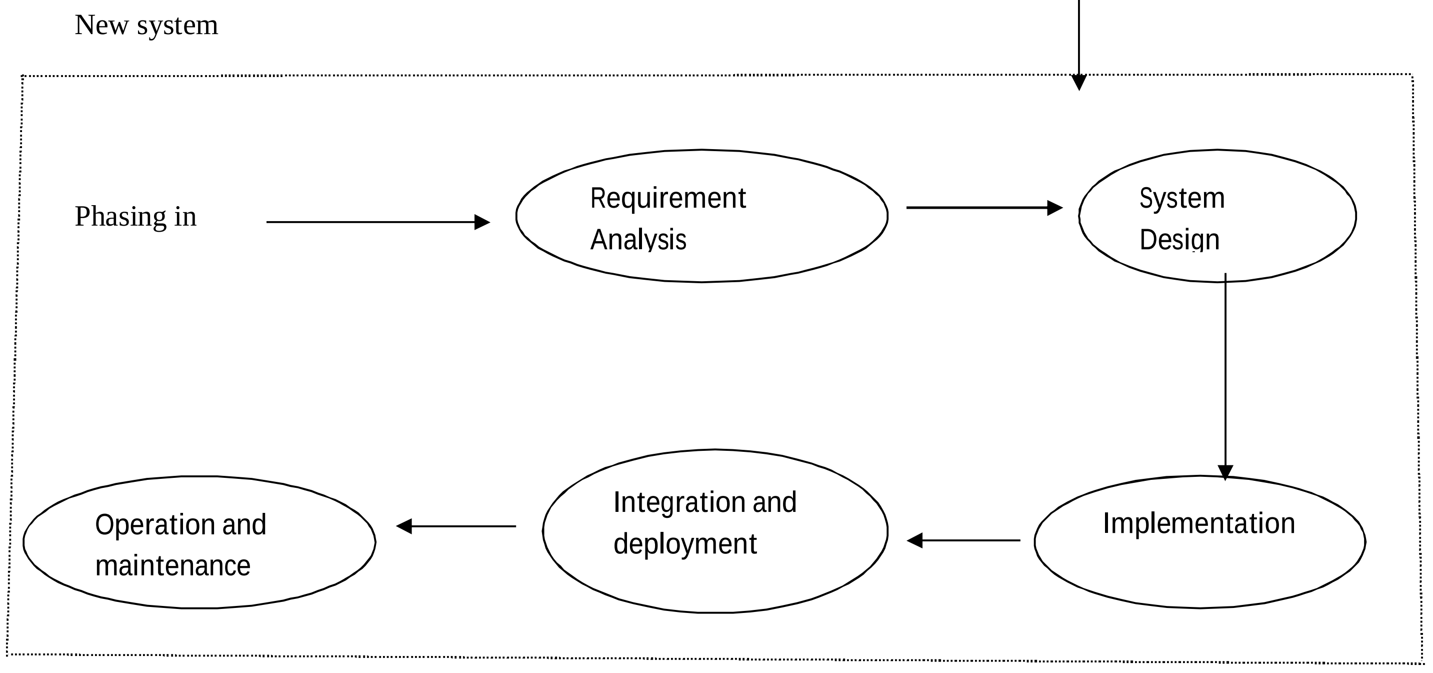

Figure 3b: Software development life cycle (Maciaszek and Liong, 2005)

In Figure 3b the phasing-out of the old system in Figure 3a becomes the phasing-in of the new system and again the same software development cycle starts for the rest of the development phases.

2.2. The waterfall process model

There are many traditional models for software development like waterfall model, incremental development, reuse oriented software engineering etc. The first model that was introduced for software development was the waterfall software development model that was introduced in the 1970 to describe the software engineering practices (Royce, 1970). According to Royce (1970) the waterfall SDLC is a sequential model for software engineering in which the software development phases are in a sequence, when one phase completed then it is documented and the same completed phase becomes the input for the next phase.

According to (Sommerville, 2011) waterfall model is a plan driven process model, in which you have to plan and schedule a specific activity before starting it. See Figure 4.

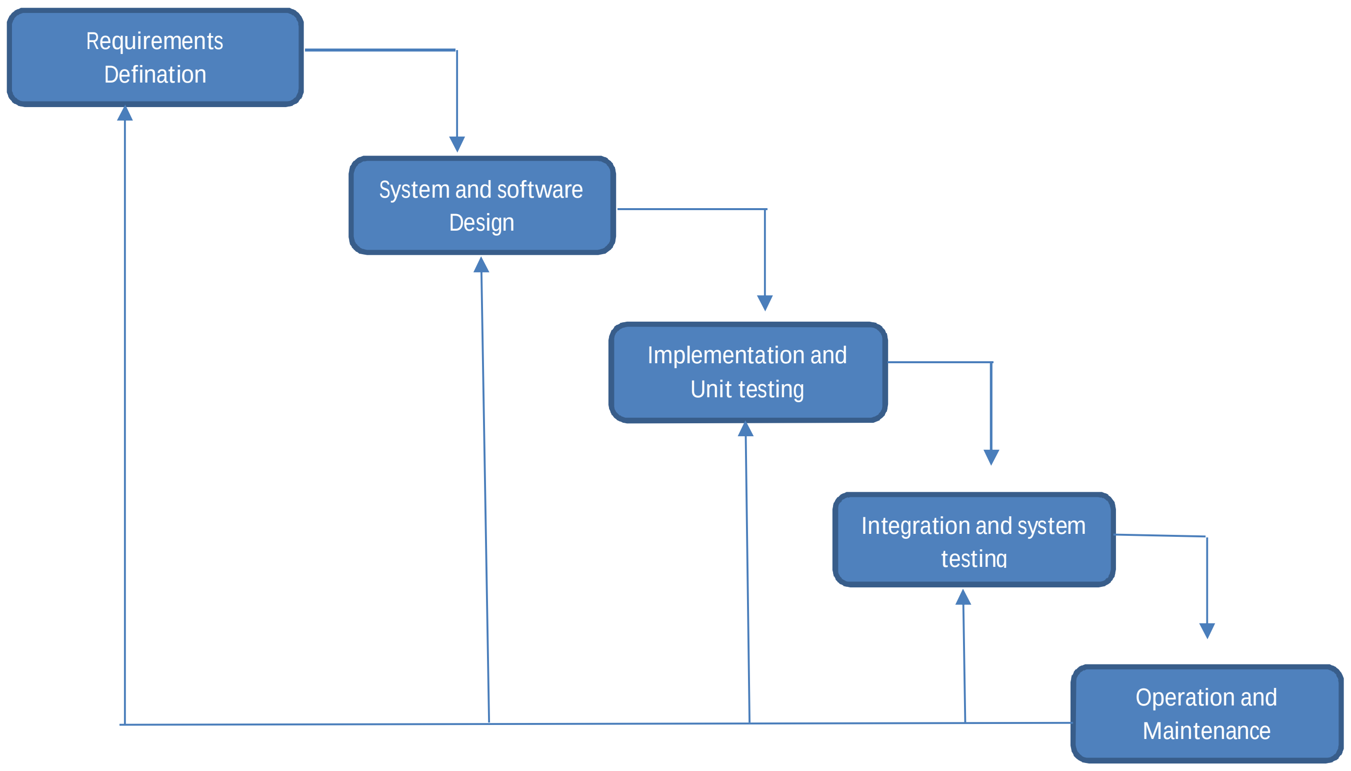
Requirements Defination
System and software Design
Implementation and Unit testing
Integration and system testing
Operation and Maintenance

Figure 4: The Waterfall model steps (Sommerville, 2011)

In the next section the steps of the Figure 4 are explained. The Figure 4 is explained by studying the book of software engineering by Sommerville (2011) and one article of Bassil (2012).

2.2.1. Requirement definition

According to (Sommerville, 2011) (Bassil, 2012) in the analysis phase which is basically called the requirement phase the software requirement specification (SRS) are defined. In this phase the behavior of the software to be developed becomes cleared. In this phase the system and business analysts declares the functional and non-functional requirements of the software. The functional requirements are usually taken out from the use cases (UC) of the software. The UC shows the interaction of the users with the system. In non-functional requirements the constraints on the requirements are defined. The non-functional requirements have no any concern with function of the software; it is concern with the properties of software like maintenance, scalability, reliability, performance etc.

2.2.2. System and software design

In the system and design phase the design of the software is created by allocating the requirements to the hardware or software system and an overall architecture of the system is established. The software architecture identifies the software abstraction and their relationship (Summerville, 2011) and according to Bassil (2012) in the design phase the algorithms are designed, the software architecture (SA) is created, database conceptual schema (CS) and logical diagram (LD) is drawn, the graphical user interface (GUI) is designed and data structure is defined.

2.2.3. Implementation and unit testing

During the implementation and unit testing phase the complete software design is converted into computer program. During this phase the whole design of the system is converted into to small components and each component is solved and then tested to work properly (Bassil, 2012).

2.2.4. Integration and system testing

During the integration and system testing phase the small components are integrated and then tested whether the components are working properly after integration. In this phase the whole system is tested to check whether the requirements that were declared during the requirement phase are meet (Summerville, 2011). This phase is also called the verification and validation (V & V) phase. The V & V is a process to check whether the software solution meet the original requirement and specification and the software is ready for the purpose for which it is produced (Bassil, 2012).

2.2.5. Operation and maintenance

The last phase of software development is the operation and maintenance phase. During this phase the delivered product is modified by finding the errors after the operation of the software. The errors are corrected the quality and performance of the software is improved (Bassil, 2012).

2.3. Advantages and Disadvantages of waterfall model

According to (Stoica et al., 2013) waterfall model has fewer advantages than disadvantages

The advantages of waterfall model are given below.

- The heavy documentation is an advantage for a new member.
- The structure design is an advantage for the new member.
- Simple and easy to use software development model.
- Each stage has an expected result which is easy to coordinate due to model rigidity.
- One stage at one time during development.
- It is recommended for clear requirements projects.

The disadvantages of waterfall model are given below.

- Some requirements may emerge after the requirements gathering phase and that create problems.
- Problems detected at a stage are not solved completely in the same stage.
- There is no any concept of changing (partitioning) the project into multiple stages.
- New requirements by the client are very expensive and cannot be adjusted in the current edition of the software product.
- Estimation of time and budget for each stage is very difficult.
- No any prototype before the finishing of the life cycle.
- Testing in the last stage of the development.
- If testing find some problem then going to the design stage is very difficult.
- Very high risk in the entire life cycle of the development.
- Not recommended for object oriented projects

In this chapter the second research question of the thesis is explained.

3. Different types of agile methods

The main purpose of software engineering is to provide the customers quality products with no defects and that can cover all the expectations of the end users. To achieve those goals several Software Development Life Cycle (SDLC) methodologies have been used. (Manjunath et al., 2013). In order to achieve the target of developing software that meets the criteria of high quality and user satisfaction the good idea is see the software requirements and the needs in the project and then select SDLC.

There has been an unstoppable advancement in the field of software development during the last few years, introducing new methodologies in this field. During the last 25 years there has been a huge development in the field of software development methods. Many software development methods have been introduced, some of those methods have been rejected, many of those methods have been accepted for some time and then replaced by some new software engineering methods, many of those methods are still valid but with some changes (Abrahamsson et al., 2002).

In 1970 Royce (1970) documented the first enhanced waterfall model (Ruparelia, 2010). But, Royce himself believed that the waterfall model is not suitable for the software development, because for a successful software development the model should have the flexibility of repetition between the phases or there should be a back and forth between the software development phases and these qualities were missing in waterfall model. Even Royce believed the iterative method will be good for software development (Kessel, 2013).

Due to the advancement in the internet field and specially the electronic business (e-business) there are huge changes in the business requirements and the traditional software development methods are not able to fulfill those changed requirements in time and within required budget and therefore they failed. Agile software development methods however have been adopted because they are very flexible to changed requirements and deliver software in quick possible time (Livermore, 2007).

3.1. Agile Software Development Alliance and Agile Manifestive

Using the traditional software development methods business software was almost late and over budget and it was not able to complete the requirements for which it was developed. In order to overcome those problems, in March 2001, 17 people from different companies met and tried to find some common grounds for software development and the outcome of that meeting results the Agile Software Development Alliance 12 principles, so called the Agile Manifestive principles as given below (Fowler and Highsmith, 2001).

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome change requirements, even late in the development. Agile processes harness change for customer's competitive advantage.
3. Deliver working software frequently, from a couple of months, with a preference for the shorter timescale.
4. Business people and developers work together daily throughout the project.
5. Build projects around motivated individuals, give them the environment and support they need and trust them to get the job done.
6. The most efficient and effective method of conveying information with and within a development team to face-to-face conversation.
7. Working software is the primary measure pf progress.
8. Agile processes promote sustainable development. The sponsors, developers and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhance agility.
10. Simplicity- the art of maximizing the amount of work not done- is essential.
11. The best architecture, requirements and designs emerge from self-organizing team.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

The purpose of the Agile Manifestive principles was the main four points. According to (Fowler and Highsmith, 2001):

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentations
- Customer collaboration over contract negotiations
- Responding to change over following a plan.
-

3.2. Agile methods

The main important issue in the agile methods is that agile methods are simple and deliver the software in quick possible time by focusing on the the most important functions first, deliver them in quick possible time and collect the feekback and reacting to that feedback (Abrahamsson et al., 2002).

According to (Abrahamsson et al., 2002) a method is said to be agile if it has the following software development properties.

- **Incremental** : small software release with rapid cycles
- **Cooperative**: customer and developers are in close communication.
- **Straightforward**: easy to learn and well documented
- **Adaptive**: that can accommodate changes at any stage in the development.

Agile methods show many benefits in different areas of software development like productivity, project visibility, software quality and many more (Lagerberg et al., 2013). Moreover in agile methods the requirements and design details are flexible and can be changes at any stage in the development phase (Lindstorm & Jeffries, 2005). Agile software development methods share many features like minimal documentation, prototyping, and iterative development (Holmstrom et al., 2006). According to (Paetsch et al., 2003) agile methods welcomes change in the requirements, continuously delivering the working software with close interaction between the business people and developers.

The agile software process has certain characteristics that make the agile software process faster (Miller, 2001). Miller (2001) proposed the following characteristics in the agile development process.

- Modularity on development process level.
- Iterative with short cycles enabling fast verifications and corrections.
- Time bound with iteration cycles from one to six weeks.
- Parsimony in development process removes all unnecessary activities.
- Adaptive with possible emergent new risks.
- Incremental process approach that allows functioning application building in small gaps.
- Convergent and incremental approach minimizes the risks.
- People oriented process
- Collaborative and communicative working style.

Based on the above discussion, few agile methods are Extreme Programming (Beck, 1999b), Scrum (Schwaber 1995;Schwaber and Beedle, 2002), Rational Unified Process (Kruchten 1996, 2000), Dynamic Systems Development Method (Stapleto, 1997), Open Source Software Development (O' Reilly, 1999), Crystal Family of Methodologies (Cockburn, 2002a), and Adaptive Software Development (Highsmith, 2000).

The thesis will consider only three agile methods (Extreme Programming, Scrum, and Rational Unified Process) from the above agile methods. These methods are outlined in the following sections 3.3., 3.4., and 3.5.

3.3. Extreme Programming (XP)

Due to the problems (long development cycles) that were raised by using the traditional development methods Beck (1999) introduces the concept of Extreme Programming after a number of successful trails on the previous XP practices (Anderson et al., 1998). Extreme Programming (XP) is a lightweight software development method that has got its popularity because of its best practices (Nawrocki et al., 2002).

XP has introduced new way of software development and is efficient, low risk, welcome changes, predictable, scientific and is different from other methods because of strong oral communication, pair programming, automated test, collective code ownership and introduced story telling culture (Beck, 1999; Juric, 2000). Figure 5 shows the XP process (Abrahamsson et al., 2002).

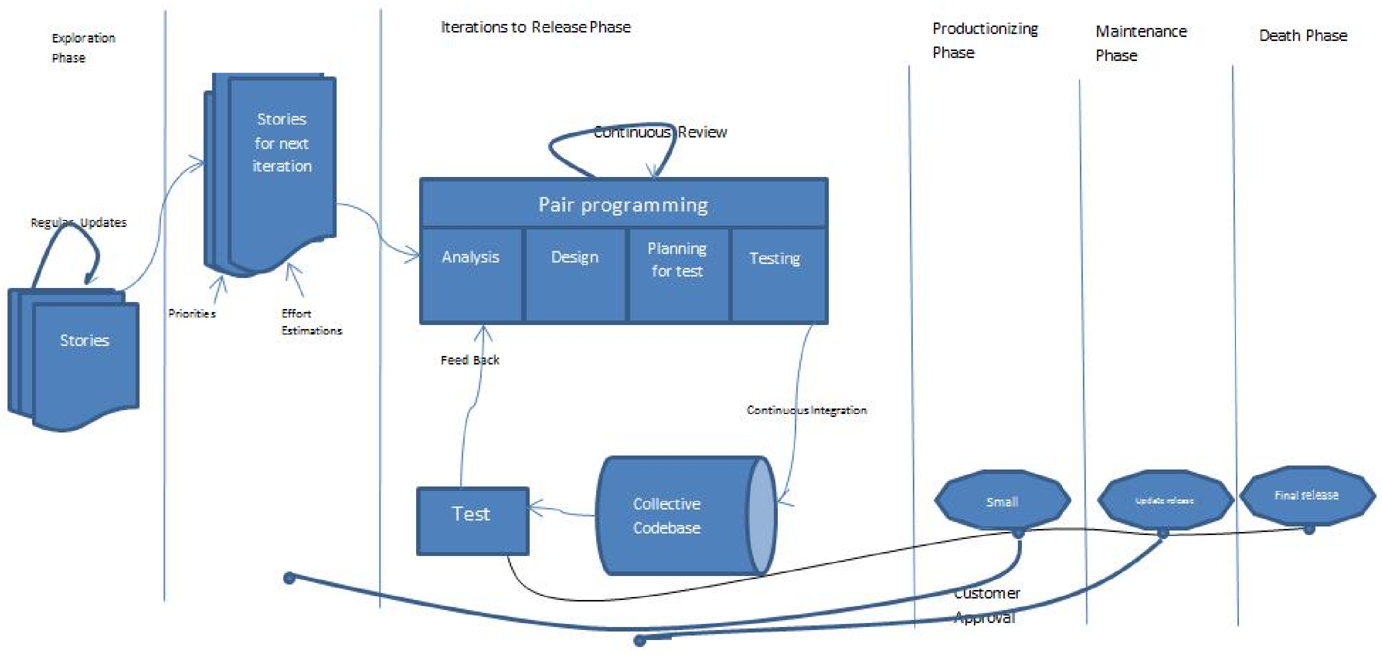

Figure 5: XP process (Abrahamsson et al., 2002).

The XP process has five phases Exploration phase, Planning phase, Iteration to release phase, Productionizing phase, Maintenance phase, and death phase. (Abrahamsson et al., 2002; Juric, 2000).

3.3.1. XP process phases

Extreme programing has five phases and has been explained below using (Abrahamsson et al., 2002) (Beck, 1999) descriptions.

Exploration phase: during the exploration phase the customers are trying to write the story cards and chose the most important stories so that they should be considered in the first release, as every story contains some specific function of the program to be built. In the same phase the project teams are selecting the tools or trying to familiarize themselves with the tools and technology that they will be using in the project. All the tools and technology will be tested and through a prototype the possible architecture of the system will be designed. The time of this phase depend on the programmer's familiarity with the technology being used. Usually the time for exploration phase is a few weeks or months.

Planning phase: during the planning phase the agreement is done for the first release by setting the priority order of the stories. The schedule and the time estimation for each story is done by the programmers. The planning phase takes about a few days and the first release takes about two months.

Iterations to Release phase: during this phase the schedule of the planning phase is broken down in to small iterations and those small iterations takes about one to four weeks. In the phase several iterations of the system are made before the first release. Each iteration consists of the analysis, design, planning for test, and the test parts. There is continuous integration after the iteration. During the first iteration the system is created and the architecture of the whole system is also created. The stories that are used in each iteration are selected by the customer. At the end of every iteration the functional test is run that is created by the customer. After the last iteration the system is ready for the production.

Productionizing phase: during this phase the before handing over the system to the customer some additional test are carried out and the performance of the system is checked. In this process, some new changes can be found and then it is checked to whether the changes can be accommodated in this current release. This process is very quick and the iteration time in this phase is very short may be one to three weeks maximum. The previous postponed ideas and suggestions are documented in this phase.

Maintenance phase: after the release of the first iteration the system is handed over to the customer. During these stage the system is in production state as well as in running phase, so some effort are required for the customer support tasks. The production speed of the system is reduced in this phase and some new members are joining the team and the team structure is changed.

Death phase: this is the last phase of the development phase. During this phase the customer is fully satisfied and has no any other story for development. In this phase the functional and nonfunctional requirements are completed. During this phase the final documentation of the system is written and there are no any further changes in the architecture, design and code. During this phase it may also be possible if the system is not delivering the required outcomes or the system is over budget.

According to Xiaohua et al. (2008) the XP method is like iteration incremental development process that provides a continuous release of iterations to customers to satisfy the customer. See Figure 6.

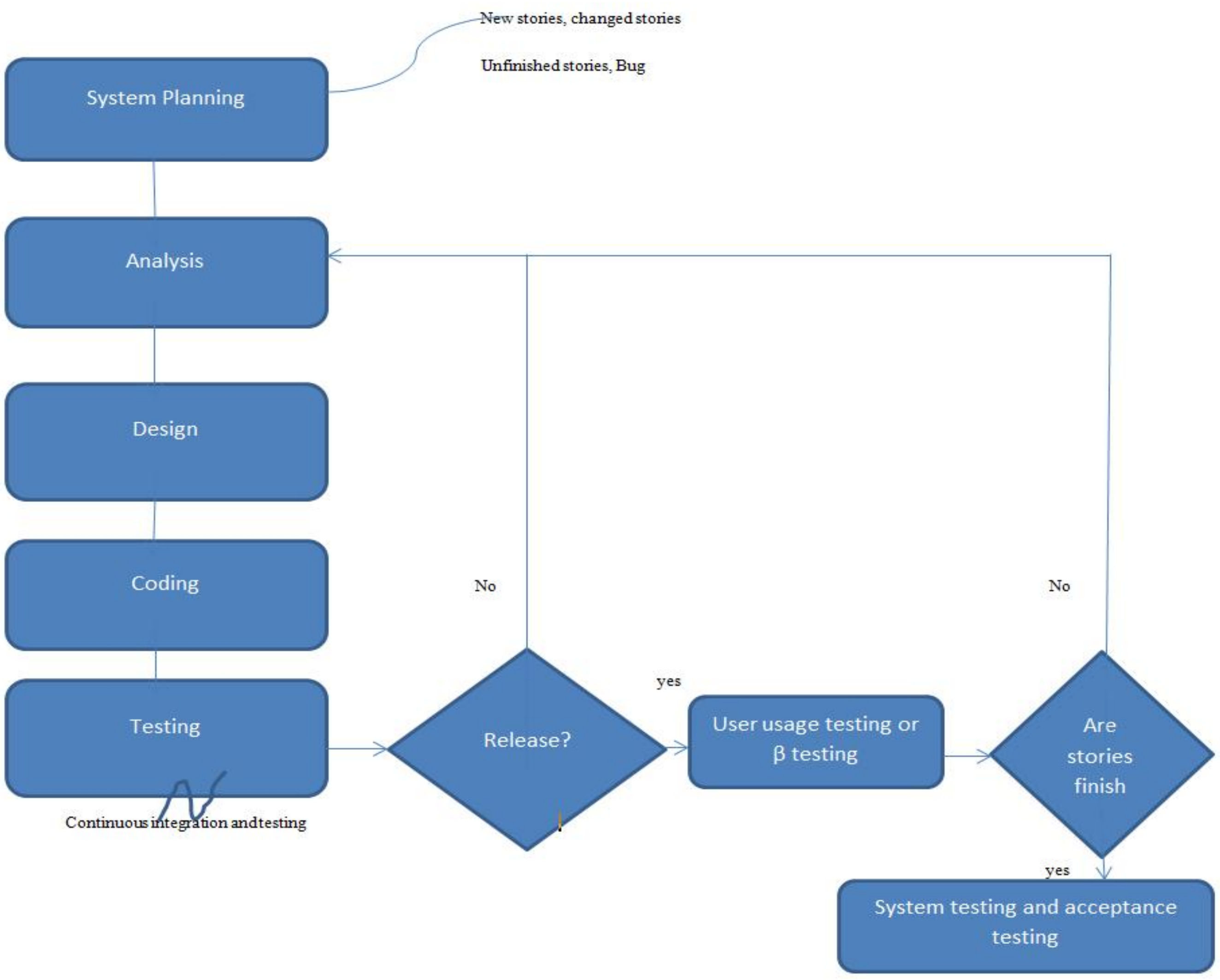
New stories, changed stories
Unfinished stories, Bug
System Planning
Analysis
Design
Coding
Testing
Continuous integration and testing
No
Release?
yes
User usage testing or β testing
No
Are stories finish
yes
System testing and acceptance testing

Figure 6: Iteration in XP development process (Xiaohua et al., 2008)

In Figure 6 Xiaohua et al. (2008) show the iteration in XP development process and the steps of how the stories are converting to working module. The process is the new story is considered and planned, the story is then analyzed and designed the coding process is completed and then testing (continues integration and testing) is carried out. If the testing result is “no” then the story is send to the analysis phase again else if testing is correct and release is “yes” then the story is send to (beta) β testing. If after the beta testing the story is not finished then story is send to analysis phase again. Else if the story is finished then the story is added to the system and the system testing and acceptance testing is carried out.

3.3.2. Roles and responsibilities in XP

There are different types of tasks in XP and for each task there is some role and these roles have some responsibilities. Beck (1999) represents the following roles and their responsibilities.

Programmer: The most important role in the XP development that writes the test and converting the customer stories to programming code. One of the success factors of XP is the close communications of the programmers and the other team members. The key responsibility of the programmer is to write the code simple and definite as possible.

Customer: Customer is the one who writes the XP stories and the functional test and gives feedback. In the functional test the customer is responsible to check that the system is completing all the requirements for which is designed. The customer is also responsible to give priority to the requirements.

Tester: customers write the functional test and the testers are helping them to write the tests. The key responsibility of the tester is to run the functional test and provide the test results. Testers also maintain the test tools.

Trackers: the role of the tracker is similar to a checker. The tracker is responsible to check the estimates made by the team during the planning. The estimates may be like the effort estimation and provides feedback about the progress the tracker. Trackers checks the iteration time with iteration plan and provide the status of the iteration and calculates that whether the project goals will be achieved within the time and resources or some change is required in the process.

Coach: the person responsible for the whole process. The coach is responsible to guide all the members of the team. Coach has very high understanding of the XP process.

Consultant: one of the external member that has some technical experience and guides the team members in some specific problem.

Manager: Manager is also called the Big Boss and he/she makes the decisions. The manager updates himself about the project from the project members. The manager has close communication with team members by trying to solve problems if any occurs.

3.3.3. XP Practices

XP has got attention in the recent years because of its best practices (Williams and Upchurch, 2001). XP practices are selected from the existing best methodologies (Beck 1999). See Table 2 below.

Table2. XP practices (Beck, 1999).

Practice	Details
Planning game	Close communication between programmer and customer. Customer decides the timing of the release and the programmer estimates the effort for story.
Short release	The productionization of the simple system in short time of about 2 to 3 weeks. The new versions are released at least every month.
Metaphor	A short description of how the system will work. The system is defined by a metaphor of set of metaphor between the customer and programmers.
Simple design	The aim is the simplest possible design for implementation at the present moment.
Testing	The unit test are run continuously before coding and the functional test being written by the customer
Refactoring	Improving the code structure while preserving its function. Removing duplications, improve communication and make the code flexible
Pair programming	At the same time two programmers are busy on one computer.
Collective ownership	The programmer are free, anyone can change code at any time.
Continuous integration	The ready code is integrated with the code base and integration test is carried out to check the code is working.
40-hours week	The maximum time for work is 40 hours in a week. Two weeks consecutive over times is not allowed. If it happens then it’s a problem to be solved.
On-site customer	The customer is available all the time for the team.
Coding standard	The programmers have to write the code according to the coding standards.
Open work space	The programming area should be large. The pair programming should be in the center of the space
Just rules	The team has some specific rules that should be followed by every team member, but the rules can be changed according to the requirement.

According to Kähkönen and Abrahamsson (2003) the XP practices can be expressed using the theoretical framework of 5-A model and their analysis shows that the possibility of explaining the XP practices using this model. See Table 3 below.

Table 3 XP practices by 5-A model (Kähkönen and Abrahamsson, 2003).

Level of Coverage	**Explained by 5-A Model**	**Areas of improvement**
Sufficient	• Continuous integration • Spike • Collective code ownership • Metaphor • Open work space • Coding standard • Refactoring • Just rules	5-A model provides sufficient coverage for practices
Explained, but need support from other theories	• Planning Game	Commitment management
	• Small releases	Business utility
	• Pair programming	Enhanced process discipline
	• On-site customer	Decision making
	• Testing	Effects of early error detection
	• Simple Design	Cost of change curve
Outside of scope	• 40-hour week	Sustainable pace

In above Table 3 the XP practices are explained using the 5-A model. In the level of coverage section, '*sufficient*' shows that the model provides full theoretical support to the XP practices like continuous integration, spike, collective code ownership, metaphor, open work space, coding standard, refactoring, and just rules. '*Explained, but need support from other theories*' indicates that the model do not provide full support to XP practices like planning game, small release, pair programming, on-site customer, testing, and simple design. Some aspects of those XP practices are short and they need some improvement, and those improvements are indicated in the 'area of improvement' section. '*Outside of scope'* section indicates that the model do not provide any practical support to XP practices like 40-hours week.

3.3.4. XP usability

According to Beck (1999) the XP method can be used in any project and there is no limit for it. But according to Cao et al. (2004) the XP method can be used only for small and medium projects that have small team structure. Hussain et al. (2008) show that most of the practices in the XP methodology can be used directly in the project while some required little changes according to the environment.

3.4. Scrum

In the recent years there has been a huge change in the software development environment, the requirements change continuously to meet the business needs during the product development life cycles and that creates too much problems for the software development teams (Rising and Janoff, 2000). According to Keenan (2004) use a well-defined software development process such as the agile methods (Scrum) can increase the delivery to be faster, high quality, and according to standards with changes during the product life cycle. During a survey conducted by Salo and Abrahamsson (2008) the agile methods like Scrum and XP are adopted in the European embedded software development organizations because of their best practices and more.

According to Schwaber and Beedle (2002) the main idea of Scrum development was managing the system development process. The scrum approach is an empirical one that is used for applying the industrial process control theory to system development for a new approach to get the ideas of flexibility, adoptability, and productivity. According to Schwaber (1995) the main purpose of Scrum is the change in the process of the system development environment and technical variables like requirements, time frame, resources and technology. This will make the development process complex and unpredictable and will allow the system development process to respond to the changes and thus will make the system development process a better one that can deliver a useful product.

Abrahamsson et al. (2002) in their study point out that Scrum provides a lot of support to existing software engineering practices (like e.g. testing in an organization). Scrum involves different management activities, and identifies deficiencies in development etc. The Scrum process is outlined in Figure 7 below (Abrahamsson et al., 2002).

The scrum process

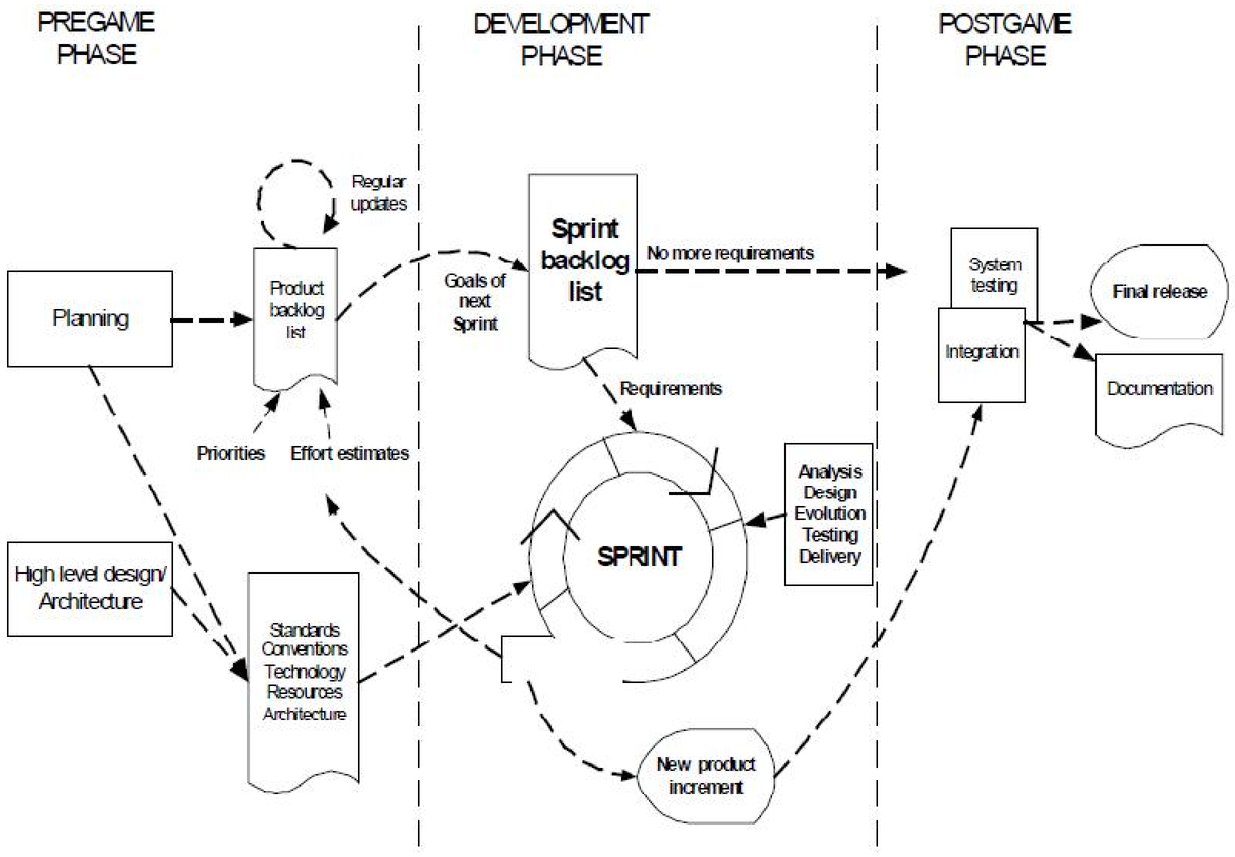

Figure 7: The Scrum Process (Abrahamsson et al., 2002).

In the next section the Scrum phases are explained in a detail manner.

3.4.1. SCRUM Phases

The scrum development process has three main phases, Pregame, Development (also called Game) and Postgame. All the Scrum phases are introduced according to Schwaber (1995) (Abrahamsson et al., 2002).

Pregame Phase: Pregame phase has two main portions, planning and architecture /high level design.

During the *planning* portion the definition of the system being developed is discussed. During this portion the product backlog list is created that contains the main overall current requirements. Furthermore the requirements are prioritized and the estimations of effort for their implementations are calculated. The delivery date and functionality of the release id estimated. The product backlog list is updated continuously (regular updates) with new requirements and priority of requirements. Moreover the planning portion also includes the definition of the team, the tools being used and other resources etc. and the verification of the approval of funding from the management.

The second portion of the pregame phase is the product *architecture / high level design.* The architecture is the high level design of the product based on the product backlog. Reviewing the product backlog items and identify the possible changes and refine the system architecture to support the new changes in the requirements and identify the possible problems arise due to the implementation of changes.

Development Phase (also called Game phase): This phase is the development phase, there is a development releases with some specific time to meet the requirements with some quality and on time with in the required budget.

The development work in Scrum is iterative. The system is developed in sprints and each sprint includes the analysis, design, evolution, testing, and delivery phases. During the development the management keeps an eye on the development, the development time, the quality of work. There are different meetings with the teams to review release plans. Different iterative sprints are carried out before the product is ready. The sprints can take about one to four weeks depends of on the complexity and risk in the product, a product may have three to eight sprints each sprint may consist of one or more teams that are carrying trying to achieve different tasks. Each sprint is review and in the review the whole team and product management are present, the review also includes customers, sales and marketing. During the review the functions of the system and its execution and all the objects assigned to the team are reviewed. There are changes in the backlog items according to the review and the team members are assigned work according to the review and also the time for the next review is calculated and that time is based on the current progress and difficulty of the present work.

Postgame (Closure): Postgame is the last phase in scrum and is executed when the requirements are completed. There are no more new product increments from the development phase. During this phase no any new item is invited and the system is ready to be released. The other activities of this phase are the integration, the whole system testing, and finally the proper documentation and the final release.

3.4.2. The Scrum Team's Roles and Responsibilities

According to Schwaber and Sutherland (2013) the scrum teams are well organized with the capability to complete their work in best way and they never depend on someone outside the team for help. The scrum teams are designed in such way that, they are flexible, creative and productive. The scrum teams deliver the products incrementally and iteratively.

According to Schwaber and Sutherland (2013), Hayata and Han (2011) the scrum team consists of Product Owner, the Development Team and Scrum Master.

The *Product Owner* is the one responsible for the increasing the product value and also the Product Owner is responsible for the Development Team work.

It is the responsibility of the Product Owner to manage the product backlog, clearly express the product backlog items, and give the order (priority) to the product backlog item best goals and achievements, the value of the work the Development Team is performing. It is also the responsibility of the Product Owner to ensure that the product backlog is clear to all, and to assure that the Development Team understands the items in the product backlog, and what the Development Team will do next (work).

The Product Owner is a single person; the role of the Product Owner is just like a committee and if anyone in the team wants some changes in the product backlog they have to contact the Product Owner. The decision of the Product Owner is final and everyone in the organization has to accept and respect it.

The *Development Team* consists of skilled professionals that are delivering the releasable increments of “Done” products at the end of each sprint. The increment can only be created by the Development Team. The Development Team is self-organized and are turning the product backlog items into increments, without the external help. The scrum team is involved in many activities like effort estimations, creating the sprint backlog, reviewing the product backlog list. Moreover with the mentioned activities the scrum team also gives suggestions about specific items that can be added or removed from the project. The size of the Development Team varies according to the project. If the team size is less than three then the interaction between the team members is decreased and because of that the productivity gain is less and also the smaller Development Team has deficiency of skills and not able to deliver the potentially reliable increment. However if the development team size is more than nine members then there is a problem in coordination between the team members and the process becomes complex. So the Development Team size should be three or a maximum of nine.

The *Scrum Master* is the one responsible for checking that the project is going on according to the scrum practices and that everyone is moving according to the scrum rules. The Scrum Master is also responsible to check the progress of the project according to the plan. The Scrum Master has a close connection to the Development Team, Customer and the Management. The Scrum Master can make necessary changes in the process to help the Development Team to increase its productivity, coaching and helping the Development Team how to achieve the task quickly and on time with high quality without depending on others outside the team. The Scrum Master also provides some services to the product owner by providing tips for the better management of product backlog.

3.4.3. Scrum usability

According to Schwaber and Sutherland (2013) Scrum is not a process or technique for software development; it's only a lightweight, simple to understand and difficult to master process framework that can be used in complex product development. But according to Rising and Janoff (2000) Scrum is a software development process that can work best with the small teams and Scrum approach is not good for large and complex team structures, but if the large project has small isolated teams then better results can be expected using Scrum.

Scrum is effective in projects with small team size that are self-organized and the team members have strong communications and collaboration (Abrahamsson et al., 2002).

3.4.4. Scrum Terminology

Scrum uses different types of terminologies during the communication in the scrum project. See Table 4 below.

Table 4: Terminologies used in Scrum Communications. (Sulemani et al., 2009)

Activity	Description
Chicken	Any one that has some interest in the project but have no any formal Scrum responsibilities and accountability.
Daily Scrum	Short meeting that every team held every day to check the work of every team member, check the work according to the schedule and plan for the next meeting.
Done	In a sprint review if something is reported as "Done" this means all the parties are mutually agreed and some thing is completed according to the organization standard and guidelines.
Estimated work remaining	This is the time when the team member estimates time left to complete the task. This estimation is carried out at the end of every day when the team members are busy on sprint backlog tasks.
Increment	At the end of a sprint when the development team develops a product with functionality and that can be shippable to the product owner stakeholders.
Sprint	Sprint is iteration or one can say repeating cycle of work that can produce an increment of the development product. The duration of the sprint is a minimum of one week and a maximum of four weeks.
Pig	A person who has made a commitment and have to fulfill the commitment of exercising one role out of the three Scrum roles Team, Scrum Master and Product Owner.
Product Backlog	List of requirements that has been given priority and allocated specific time for completion, the list can be changed according to business needs, or technology changes etc.
Product Backlog Items	Contains the functional and non-functional requirements, issues that have been given priority according to the business needs and dependencies and is estimated. The precision of estimations depends of the priority of the Product Backlog Item.

Activity	Description
Product Owner	A person that is responsible for the Product Backlog and to increase the value of the project. The Product Owner is the person responsible for making the project and its resulting product interesting to everyone in the team.
Scrum	A mechanism in the game of rugby for getting an out-of-play ball back into play.
Scrum Master	A person who is responsible for the proper implementation of the Scrum process and utilization of its maximum benefits.
Sprint Backlog	It contains a list of tasks for completing a Sprint. Each of the task has some specific time and someone in the team is responsible for the task.
Sprint Backlog Task	A task defined by the team member to turn the Product Backlog items into a functional product.
Sprint Planning Meeting	It’s a one day meeting of about eight hours before a four weeks sprint. In this meeting the planning and estimations for the whole Sprint is carried out. The meeting is carried in two portions each of four hours. In the first four hours the Product Owner presents the priority of the Product backlog and in the second four hours the Team plans how to complete the sprint.
Sprint Retrospective Meeting	It’s about three hours meeting facilitated by the Scrum Master at the end of the sprint and it is decided what should be changed or removed so that the next sprint should be productive.
Sprint Review Meeting	It’s a four hours meeting at the end of each Sprint where the team collaboration with Product Owner and the stakeholders and discuss about the Completed Sprint after the demonstration of the Product Backlog items. It is also discussed of what should be done for the next Sprint.
Stakeholder	Anyone affected from the project is a stakeholder, may be someone interested in the outcome of the project,
Team	A well organized group of skilled people responsible for the managing themselves to develop the product successfully and complete all the Sprints.
Time Box	It’s a specific period of time that cannot be extended and the work is to be completed within that specific time.

The above all terminologies have been used in the communication between the Scrum team members. All the members of the Scrum team are experienced and every one can understand the terminologies very easily instead of the simple language. Using those terminologies in communication can reduce time and make understanding better.

3.5. Rational Unified Process (RUP)

Rational Unified Process (RUP) is a Software engineering process framework that considers the best practices of the software development that are used in many projects and organizations. RUP uses the Unified Modeling Language (UML) for different models that are used in the development process. The software organizations can use RUP according to their requirements (Kruchten, 2002).

According to Ambler (2014) many people think that there is a huge difference between agile and RUP, but that is totally wrong. RUP can be as much agile as you want to you use it. If RUP can be used as agile there are some points that can be considered. Iteration period should be short up to four weeks. Focus should on the working software. Use such techniques that can make the software of high quality, like testing first before development, coding conventions and refactoring. There should be high communication and collaboration as much as possible within the team and also with the stakeholders. Involve the business stakeholders, data professionals and quality assurance professional during development. Streamline RUP as much as possible.

According to Del Maschi et al. (2007) RUP uses a discipline manner to establish tasks and assign responsibilities within the development process. The main aim of the RUP is to satisfy the customer with a high quality software product within time and according to the budget.

In another study Monteiro et al. (2012), Jacobson et al. (1999), and Kruchten (2002) define RUP as an iterative development process in which every member of the organization has a task and responsibilities in order to achieve a high quality product within the specific time and budget and that can satisfy the customer by fulfilling the maximum needs of the customer. According to Monteiro et al. (2012) the RUP framework consists of three main components: activities, roles and artifacts. Every project has all these three components in different manners depend on the nature of the project. Every project has a group of actors and each actor in a project is performing one or multiple roles, each role in the project is participating in certain activity or activities and as a result of these activities one or more artifacts are produced. In RUP software development process there are more than eighty artifacts, one hundred and fifty activities and round about forty roles.

3.5.1. The Rup Process Phases

According to (Rational Unified Process, 1998), Guo et al. (2011) with the time dimension the RUP can be divided into four phases *Inception*, *Elaboration*, *Construction* and *Transition*. The completion of each phase gives up a major milestone. The next phase only starts when the previous phase completed successfully.

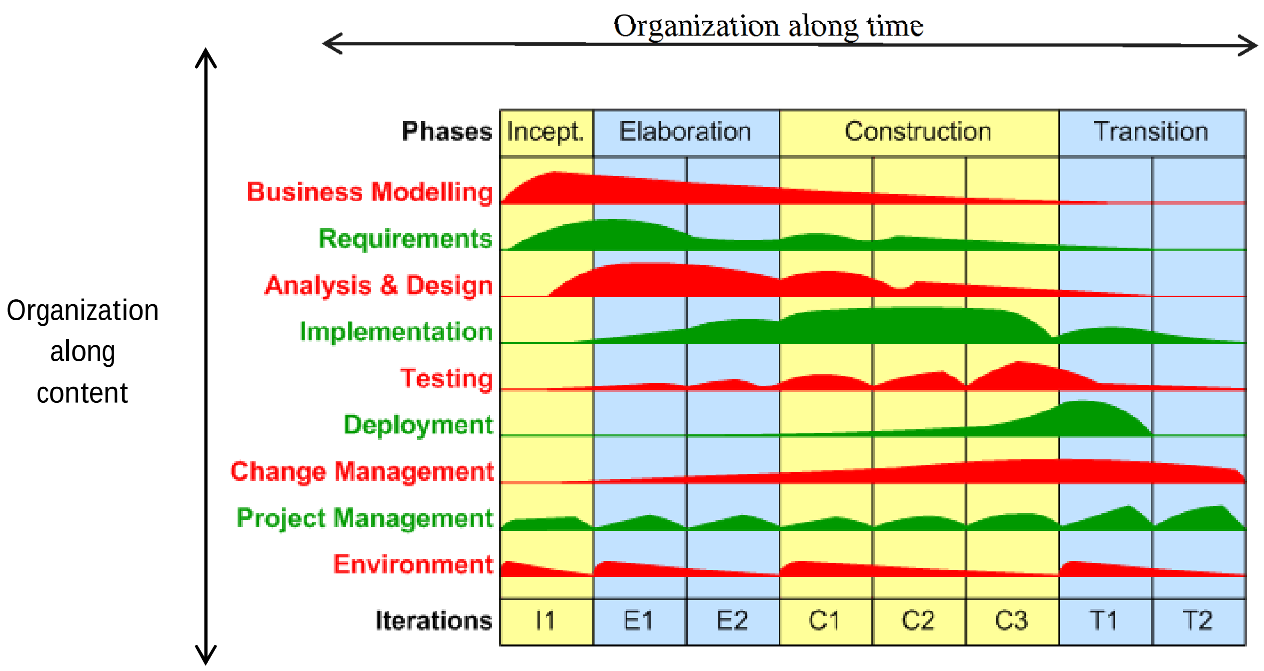

Figure 8: RUP phases and workflows (Rational Unified Process, 1998)

The below RUP phases are explained using the Rational Unified Process (1998).

The four phases of RUP are Inception Phase, Elaboration Phase, Construction Phase, and Transition Phase.

During the *Inception* phase the business case for the project is established that includes the success criteria, the risk assessment, estimation of resources and the phase plan that's shows the dates of the major milestones of the project. During the inception phase the project scope is also defined. Here in this phase the actors are identified and their interaction to the system is defined.

Outcomes of Inception phase are Vision document, Initial use case model (20 % complete), Initial business case, Initial risk plan, Project plan, and one or many prototypes.

The milestone of the Inception phase is the *Lifecycle objectives*.

One of the important phases is the *Elaboration phase*. During this phase the entire architecture of the system is developed. The problem domain is analyzed and a project plan is developed. Here in this phase the most use cases and the actors are identified and described. The architecture of the system is developed and the prototype is created. During this phase the requirements, architecture and project plan is assured to be stable enough.

Outcomes of Elaboration phase are Use case model (80% complete), Requirements, Software architecture description, Architecture prototype (executable), Revise business case and risk list, Project plan, User manual.

The major milestone of elaboration phase is *lifecycle architecture*.

The third phase is the *Construction phase* in which all the remaining application features and remaining components are developed and integrated into the system and tested. This phase is also called the manufacturing process. During this phase the main emphasis is managing the resources, cost of the product and quality of the product and the time schedule of the product. During this phase the tests (alpha, beta and other tests) speed is kept as fast as possible.

Outcomes of Construction phase are Integrated product, User manuals, and Current release documentation.

The milestone of Construction phase is *Initial Operational Capability*.

The fourth phase is the *Transition phase*. That is the phase in which the product is mature enough to be hand over to the end users, based on the demands of the user new releases are developed after removing the problems, bugs etc. the transition phase consist of beta testing, piloting and the training the maintenance team and users and releasing the product into market.

Outcome of transition phase is the *Final release of the product.*

The major milestone of the Transition phase is the *Final Product*.

3.5.2. RUP Workflow Dimensions

According to Rational Unified Process (1998), Guo et al. (2011) when RUP is considered according to workflow dimension then there are nine core workflow dimensions out of which six are core engineering workflows and the three are supporting workflows.

In Figure 8, the six core engineering workflows are Business modeling workflow, Requirement workflow, Analysis & Design workflow, Implementation workflow, Test workflow, and Deployment workflow.

The core supporting three workflows are Project management workflow, Configuration and change Management workflow, and Environment workflow.

One of the best core workflow in RUP is the *Business Modeling* in which the business engineers and the software engineers communicate which each other through a common language (use cases) and there is traceability between the business and software models. Business use cases are documented in the Business Modeling and the output of the business engineering is properly used as an input to the software development.

The main theme of the *Requirement Workflow* is to explain what the system will do and how the customer and the developers agree on the requirements. The requirements (functional requirements) are elicited through the use cases; those requirements are then organized and documented. Non-functional are explained in the supplementary specifications.

During the *Analysis and Design workflow* the realization of the system during the implementation phase is showed that how the system will perform in specific environment, how the system will fulfill the requirements, how the system should be flexible to accommodate the changes in the functional requirements. The design model is created in the Analysis and Design workflow.

In the *Implementation workflow* the code is organized in layers in terms of subsystems and the objects and classes are implemented in terms of components and then tested and the individual work of the members (teams) are integrated and checked as an executable system. One of the benefits of the RUP is the reusability of the existing components with the new components and it makes the system maintenance easy and increases the chances of the reuse for future purpose.

During the iterative approach the testing process continues throughout the project, and that is why the defects and problems can be fined early and easily and it saves the cost of fixing the defects in the later stage. The tests are reliability test, functionality test and performance test (application and system). In the *Test workflow* the interaction of the objects are verified and the integration of the all the components are verified that they are working properly as a system. The requirements for which the system is developed are tested and checked that they are implemented correctly. During the testing workflow the defects are identified and ensure before the deployment of the software.

The *Deployment workflow* has less details as compared to the other workflows as in the deployment workflow the software developed successfully is handover to the end user. During the deployment workflow the software is released externally. The main activities in the deployment workflow are packing and distributing of the software, its installations and help to the users. In some cases during the deployment workflow the beta tests are planned and conducted.

The *Project management workflow* addressed some main aspects of the iterative development process. The main activities of the project management workflow is to make a framework for the software projects, providing the guidelines for planning of the project, the staff required for the project, how to execute the project and how to handle and monitor the projects and how to manage the risks during the project.

Various artifacts are produced by the team members in the project, the *Configuration and change management workflow* describes how to control those artifacts developed by many people in order to avoid the confusion that may be too much costly and to ensure that there should be no any conflict in those artifacts during the Simultaneous update, Limited notification and Multiple versions. During this workflow it is described how to work in parallel development, development done at multiple sites, and how to automate the building process. In the configuration and change management workflow there is an audit on why, when and by whom an artifact is changed.

The *Environmental workflow* describes which development environment should be selected and provide both the tools and processes for the development team. The environmental workflow also provides a Development Kit (DK) that contains the guidelines, tools and templates that are essential for the process.

There is much iteration in the three phases (elaboration, construction, and transition) of RUP. Each of these phases may have two or more iterations for its completion.

3.5.3. Best Practices of RUP

According to Rational Unified Process (1998) RUP uses some of the best proven approaches for the software development and these best approaches are called the best practices. The RUP provides many facilities for the each member of the software development team including tools, guidelines, templates etc.

To increase the success rate of the software project the software industry figures out some practices that can be used to get high quality, cost effective reliable within time projects. They give those practices the name best practices (Ahmad et al. 2008; Kruchten, 2000).

There are six best practices for software development which are Develop Iteratively, Component based architecture, Verify quality, Control Changes, Manage Requirement, and Visual modeling.

Agile software development is *Iterative based*; there are several iterations within a project in agile development process. The main idea in iterative development is design short, build less and test less and at the same time the developer and the customer can monitor the progress very easily in each iteration. The *Component based architecture* mainly separates the functionality and reusability. During component based architecture the system is divided into many sub system and each sub system defines core functionality. The *Quality* of the system is verified by verifying and validating the functional and non-functional requirements of the system under consideration. The quality of the system if verified in each iteration. During each iteration the problems in requirements, design and implementation are identified in early stages and corrected. During the iterative development the requirements can be changes at any stage during iteration the *Control change* tracks and monitor the changes because of the change requirements and reallocate resources, managing the requirements and changes in the design iterations and planning. The *Manage Requirements* explains how to get the requirements, organize those requirements and document those requirements (functional and non-functional) and constraints on those requirements. The *Visual modeling* provides the graphical representation and behavior of the architecture and the system components. The visual representation of the system is also helpful in elaboration of requirements and the system functionality.

3.5.4. The roles and responsibilities in RUP

RUP is an iterative development process that can assign different tasks and responsibilities within organization. These roles and responsibilities depend on the projects being under consideration and organizations. Different actors are performing one or many roles in the project. About the roles and responsibilities in RUP, many researchers have different ideas.

In a research Abrahamsson et al. (2002) the RUP assign roles on activity basis and there are about thirty roles. The main roles are Designer, Architect, Configuration Manager, Design Reviewer etc.

During another research (Monteiro et al., 2012) the RUP have about 39 roles that can divide into two categories essential roles (13 roles) and non-essential roles (26 roles). These non-essential roles do not mean that they are not important but they can be reduced and cannot be eliminated. These roles can also be merged. In Figure 9 the comparison between the roles in RUP are shown (Monteiro et al., 2012).

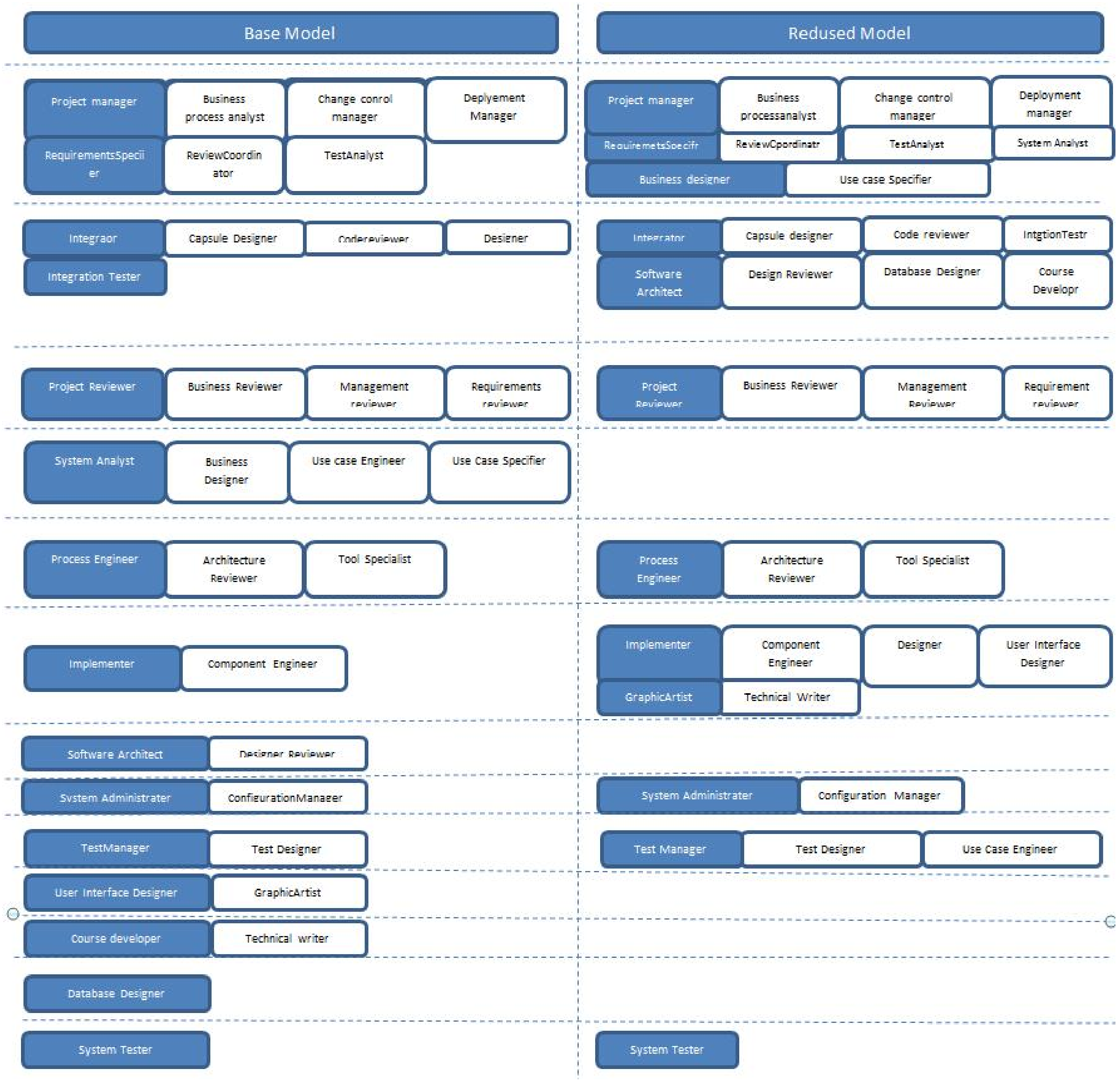

Figure 9: Comparison between Roles in RUP (Monteiro et al., 2012).

In Figure 9 Monteiro et al. (2012) show the comparison between the two models base model that contains all the roles of the RUP team and reduced model in which some RUP team roles have been eliminated. In the reduced model the roles of *system analyst, software architect, user-interface designer, course developer and database designer* have been eliminated from the reduced model. The responsibility of those roles is mapped to the other remaining roles.

3.5.5. RUP usability

The Rational Unified Process can be adopted as a whole or in parts in projects by the organizations. In many cases RUP can be used in many organizations with a small change in its process (modification) (Kruchten, 2000).

The RUP can be used in small projects with short product cycles and also can be used in large projects with large teams and long product cycles. RUP process is successful in any project whether small or large (IMB white paper, 2003).

In this chapter the third and fourth research questions of the thesis are explained.

4. Agile methods in Software Development

Due to the heavy growth of electronic commerce (e-commerce) and internet availability the success of every organization depends on the technology (software) being used for their business process and business needs. The technology is growing at a very high rate in the growing competitive market. The high competition between the organizations has created a challenge for the software companies to deliver the software on time and within budget. But delivering the software on time and within budget is difficult for the software companies because the business organizations requirements are changing continuously due to change in the business process.

The traditional software development models are not flexible to changed requirements resulting in delay of the software and also the software become over budget because any change in the requirements at the later stage of software development is very expensive using traditional software models (Stoica et al., 2013).

To solve those problems of software development and to deliver the software product on time and within budget the software companies needs some software development models that can easily accommodate changes at any stage of software development. Agile methods have been introduced that can be used to reduce those problem. According to (Abrahamsson et al., 2002) and (Lindstorm and Jeffries, 2005) agile methods are very simple and deliver the software in short time by focusing on the most important functions first deliver those functions quickly, collect feedback from them and react to those feedback. In agile methods the requirements are very flexible and can accommodate changes in the requirements at any stage of the development phase.

4.1. Difference between Traditional and Agile software development

According to (Stoica et al., 2013) there are many differences between the Agile development and the traditional software development. The traditional development use the predictive approach for software development and agile development uses adaptive software development methods. In traditional software development the team moves with a predefine plan for different tasks and activities for the whole system development life cycle (SDLC) while in agile methods team uses the plan for a specific task. In traditional software development the requirements are gathered at the initial stage of software development and changes at the later stage of SDLC are very difficult and expensive, while in agile methods the requirements are changing at any stage of the development phase.

The detailed differences between traditional and agile software development are given below in Table 5. (Bohem, 2002) (Stoica et al., 2013)(Awad, 2005).

Table 5: Difference between agile and traditional software development.

Activity	Traditional Development	Agile Development
Fundamental Hypothesis	Complete specifiable systems, predictable and developed with detailed planning.	Software quality is high, developed by small teams by using the principle of improvement in design, getting feedback and testing after feedback and make change.
Management Style	Command and control	Leadership and Collaboration
Knowledge Management	Explicit	Tacit
Communication	Formal	Informal
Development Model	Life Cycle model (Waterfall, spiral etc.)	Evolutionary-delivery model
Organizational structure	Bureaucratic, high formalized, targeting large organization	Flexible and participative, encourages social cooperation, targeting small and medium organizations.
Quality control	Difficult planning and strict control. Difficult and late testing.	Permanent control on requirements, design and solutions, permanent testing
User requirements	Detailed and defined before coding	Interactive input
Cost of restart	High	Low
Development Direction	Fixed	Easily changeable
Testing	After coding is complete	Every iteration
Client involvement	Low	High
Additional abilities required from developers Activity	Nothing in particular Traditional Development	Interpersonal abilities and basic knowledge of the business Agile Development
Scale of project	Large scale	Low and medium scale
Developers	Oriented on plan, with some adequate abilities, access to external knowledge	Agile, with advanced knowledge , co-located and cooperative
Clients	With access to knowledge, cooperative, representative and empowered	Dedicated, knowledgeable, cooperative, representative and empowered
Requirements	Very stable, known in advance	Emergent, with rapid changes
Architecture	Design for current and predictable requirements	Design for current requirements
Remodeling	Expensive	Not Expensive
Team Size	Large team size	Small team size
Project size	Large project size	Small project size
Primary objectives	High Safety	Quick value

4.2. Project success or failure rate of Traditional and Agile projects

The success or failure of project depends on certain organizational success criteria, a project is considered to be successful if it is delivered on time and it meets the success criteria of the organization. A project is said to be challenged if it is delivered on time but it's not according to the success criteria range of the organization; a project is said to be failed if the team is not able to deliver the software solution (Dobb, 2010). In Table 6 success and failure rates of agile and traditional projects are presented (Dobb, 2010).

Table 6: Success and failure rates of Agile and Traditional projects (Dobb, 2010).

Projects	Successful	Challenged	Fail
Agile	60 %	28 %	12 %
Traditional	47 %	36 %	17 %

Table 6 shows that the success rate of agile projects is better than traditional projects. The rate of success of agile projects is 60% and that of traditional projects is 47%. Project failure rate of traditional projects is higher than agile projects. The failure rate of traditional projects is 17% and the failure rate of agile methods is 12%. About 36% of the traditional projects are challenged, which is higher than that of agile projects which is about 28%.

Based on Rico (2013) agile projects success rate is much higher than the traditional software projects. According to Rico(2013) the success rate of agile projects are about three times more than the traditional projects and the failure rate of Agile projects is less than three times than the traditional projects. (See Table 7).

Table 7: Comparison rate of Agile and Traditional projects (Rico, 2013).

Projects	Successful	Challenged	Fail
Agile	42 %	49 %	9 %
Traditional	14 %	57 %	29%

According to Rico (2013) the rate of failure of global projects is decreasing as the competition in the business area is increasing. Table 8 shows ten years survey on global projects success, failure, and challenged mentioned by Rico (2013).

Table 8: Success and Failure of Global Projects by (Rico, 2013).

Year	Success	Challenged	Failure
2010	33 %	41 %	26 %
2008	32 %	44 %	24%
2006	35%	46%	19%
2004	29%	53%	18%
2002	34%	51%	15%
2000	28%	49%	23%
1998	26%	46%	28%
1996	27%	33%	40%
1994	16%	53%	31%

According to Rico (2013) the main reason of the project failure is because of the too much requirements in traditional projects. As shown in Figure 10 about 7% of the requirements are always used, 13% of the requirements are often used and 19% of the requirements are rarely used, around 16% of the requirements are sometimes used and about 65% of the requirements are never used at all. (See Figure 10)

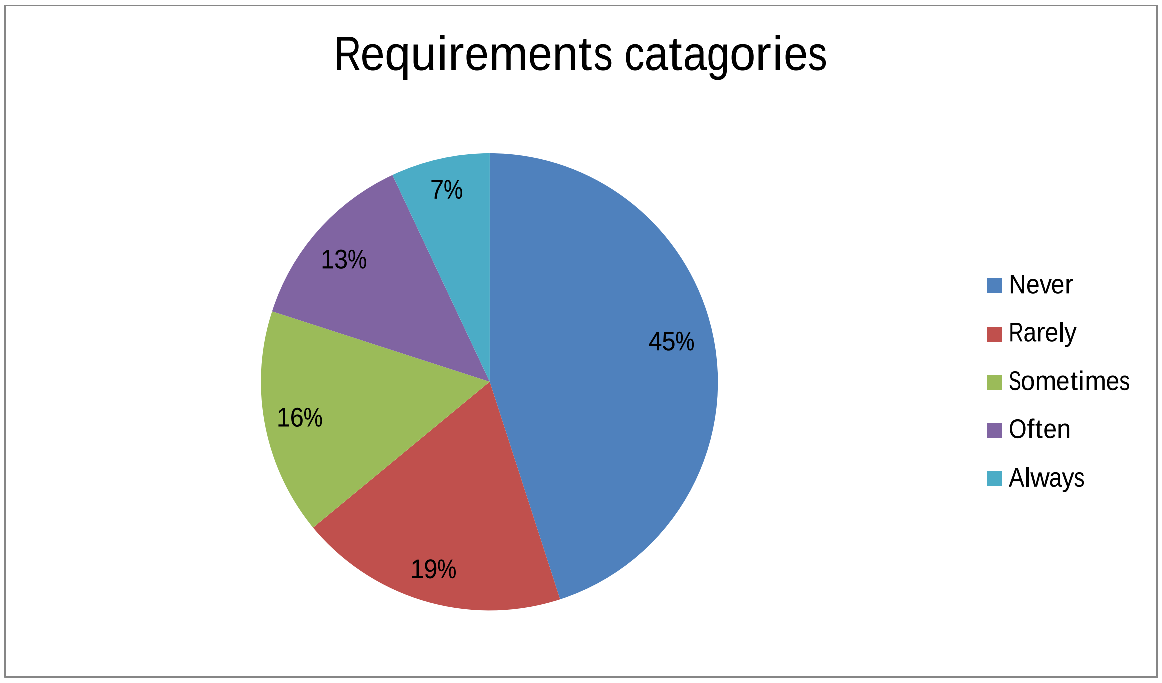

Figure 10: Requirement categories Rico (2013).

Kumar (2009) claims that the success rate of agile methods in higher than other software development methods. See Figure 11.

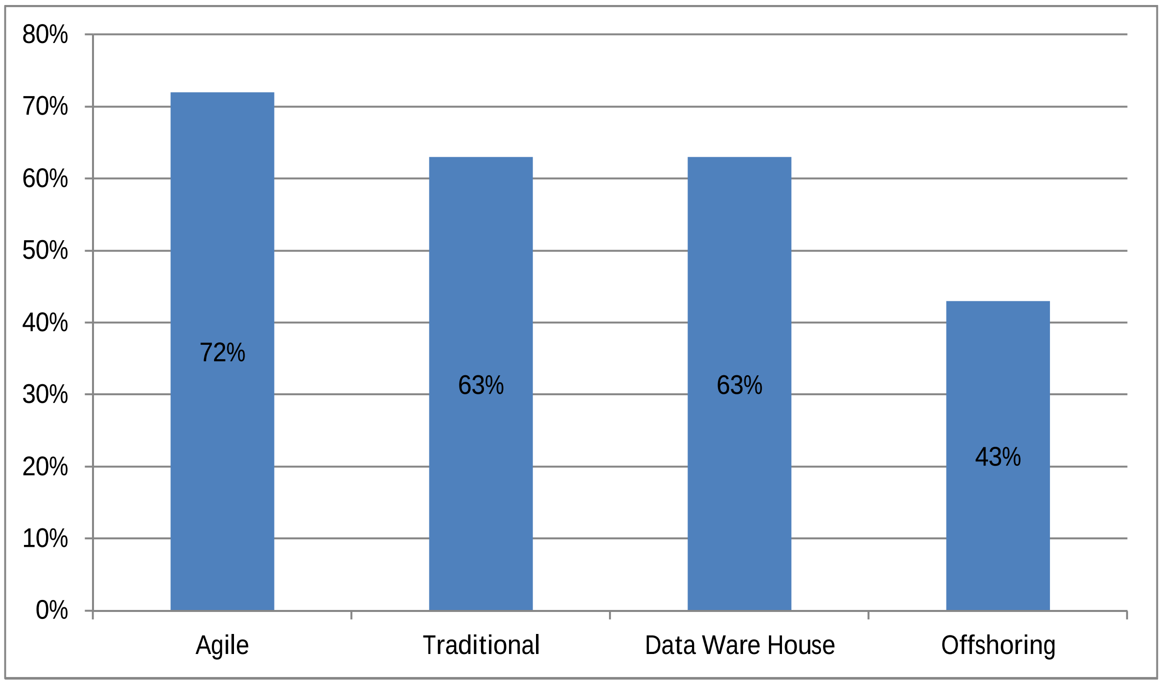

Figure 11: Agile project success rate Kumar (2009).

According to Kumar (2009) the success rate of agile projects is about 72% and that of traditional projects is about 63%. The success rate of data ware house projects is 63% and that of offshoring projects is 43%.

4.3. Success Factors in Agile Projects

The agile methods play an important role in the success of the software projects in this competitive business environment. The success of agile methods is as a result of some factors that were either not used in the previous software development methods (traditional methods) or used but not with care.

Misra et al. (2009) point out some factors for the success of agile software development. *Reduced delivery Schedules, increased return on investment (ROI), increased ability to meet with the current customer requirements, increased flexibility to meet with the changing customer requirements, improved business process*.

According to Misra et al. (2009) the above success factors reflects some areas of success of any project, such as *reduced time, reduced cost, and increased quality*. The success factors are divided into two parts, organizational factors and people factors. The hypothetical success factors are given below in Figure 12 (Misra et al., 2009).

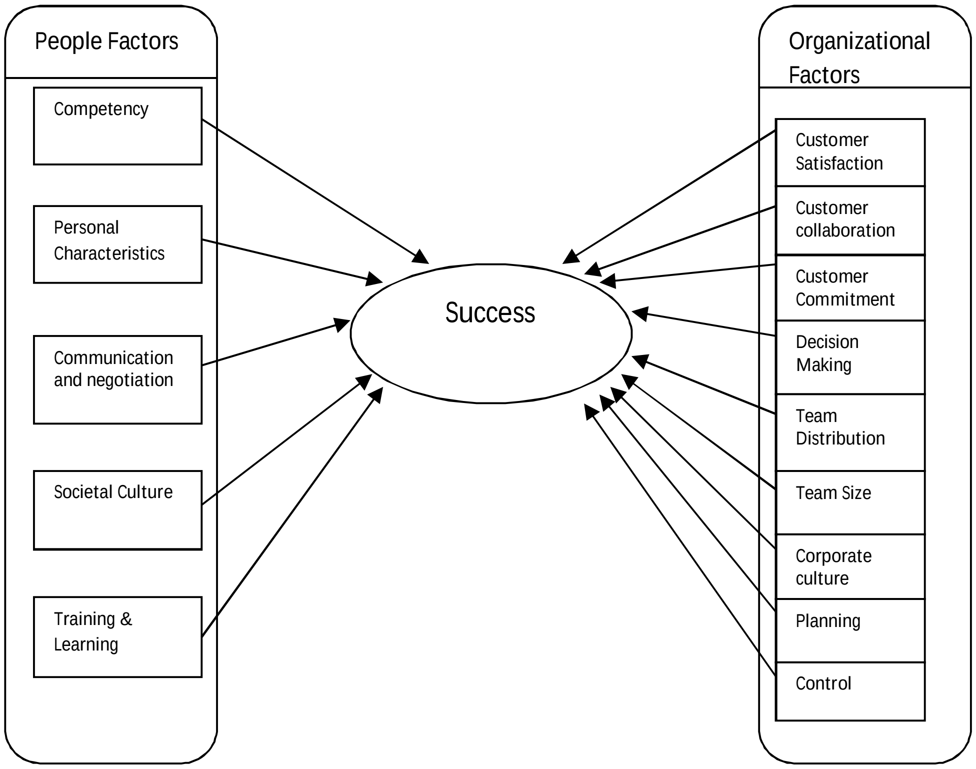

Figure 12: Hypothesized success factors Misra et al. (2009)

According to Misra et al. (2009) the organizational factors consist of *Customer centric issues.* The customer centric issues include customer satisfaction, customer collaboration, and customer commitment. Customer collaboration and Customer satisfaction are the main goal of agile software development. The customer is available throughout the development process. Customer is always active and motivated in the development process and this customer commitment is one of the best success factors in agile software development. The *Decision time* in agile development process is short as the development teams are normally free to take their decision by their own. Because of the customer collaboration there is close communication in the team, the team members take the decisions by themselves in short possible time and get success. *Team distribution* in agile software development is centralized. In centralized team structure there is no communication gap between the team members and they can discuss the matters easily. Centralized team structure is one of the best success factors in agile methods. *Team Size* in agile methods is small and communication between the small team is very easy and simple. If the project is large then the team member are large in that situation the team is divided in to multiple small teams so that the communication become easy. Small team size in project is one of the success factors in agile software development.

Due to *Corporate culture* in agile software development there is a collaboration of customer and customer feedback is very important, the organizational culture should be adaptive to changes. The corporate culture is one of the best success factors in agile methods. *Planning and control is* one of the most important success factor in agile methods is the planning and control. In agile methods there is planning for every activity in the team meeting and that plan is followed and controlled very carefully.

According to Misra et al. (2009) the people factors include *Competency* in agile methods. The team members are experienced in their fields and have enough knowledge about the tools that they are using in the project. The team members have good interpersonal and communication skills and that's one of the key factors of the success of agile software development. *Personal characteristics* in agile methods the team members are not only experienced in their field but they also have strong personal characteristics such as honesty, responsible, friendly attitude, give equal importance to all team members, work with other team members easily. *Communication and negotiation is o*ne of the most important success factors of agile software development. In agile software development the effective communication between the team members and between the customer and the team members are very fruitful. In agile methods there is close communication between the developers, operations, support, customer, management. *Societal culture* in agile software development is almost similar to the development of other products in a culture, in which the organization is working. If the people in the culture in which organization is working are motivated and the individuals have the habit to learn from each other, they are honest, collaborative and take the responsibility of work. Working in such a culture is very fruitful. *Training and learning* in agile software development is very less formal and the individuals in the teams have the eager to learn from different members in the team and share information with each other. The team members are getting experience from other members in the team by sharing information with each other's.

Beside the above all success factors there are some other success factors that can also be considered in agile software development. All those other factors are discussed below in sections 4.3.1, 4.3.2, 4.3.3, and 4.3.4.

4.3.1. Requirements and Customer involvement

In Agile methods the requirements are gathered at a regular interval and the requirements are giving priorities and also the requirements can be changed at any stage in the development phase. In agile methods the requirements are gathered by involving the customer, the customer is answering all the questions of the developers and is empowered to make decisions. The involvement of the customer in the whole software development phases is one of the best reasons of the success of agile projects (Paetsch et al., 2003). Agile development always welcomes the changed requirements due to the change in the business process and accepts the changes at any stage in the development phase (Livermore, 2007) and due to the iterative development in agile methods the responding time of the new requirements and the changed requirements is very short and can be adjusted by reprioritized the new requirement or changed requirements in the next iteration (Wang et al., 2009). Due to the high integration of the customer in the development process the requirements are easily adjusted in agile methods and because of that there is a positive relationship between the customer and developers and the customer satisfaction is very high which leads to the success of the project (Kohlbacher, 2011).

4.3.2. Close communication

In agile methods the project team size is small and they are in close contact with the customer. So there is a close communication between the team members, and the customer and the team members and they can easily share their ideas and point of view. The active communication between the team members and also with the customer is very useful in building trust (Hasnain, et al., 2013) and discipline amount the team and the customer and resulting the highest customer satisfaction. In agile methods there is a constant communication in the entire iteration because of the changing requirement during each iteration. It reduced the project cost and time (Bhalerao, 2010; Hanakawa et al., 2004).

Agile communication cycle consist of information gathering, planning, developing, testing and delivering the running software to customer, every phase require close communication. The agile iterative cycle consist of three sub phases primary, mid and end phase (Bhalerao, 2010). See Figure 13.

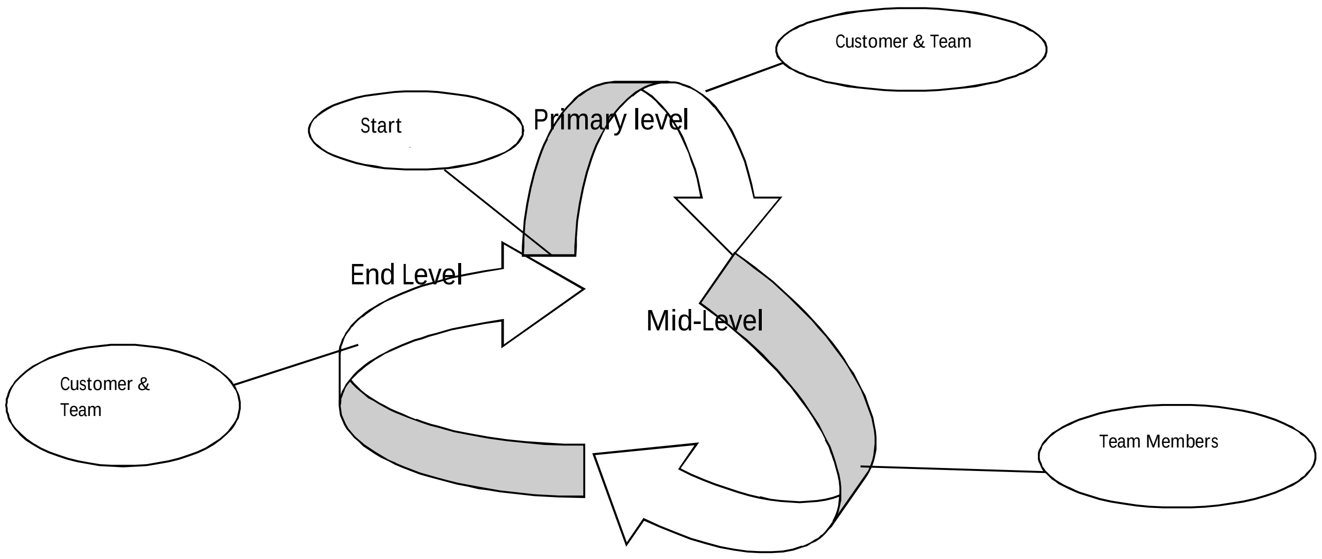

Figure 13: Agile communication cycle (Bhalerao, 2010).

According to Bhalerao (2010) in Figure 13 during the primary level of communication the customer and the team members communicate with each other, the customer gives information about the project requirements. In the Mid-level the team members gather meeting with each other and discuss about the requirements. At the End-level the customer and the development team start communication again. The development team discusses the requirements with the customer.

In another study Korkala et al. (2006) the main importance in communication is given to the face-to-face communication at white board and face-to-face conversation in agile methods. See Figure 14 below.

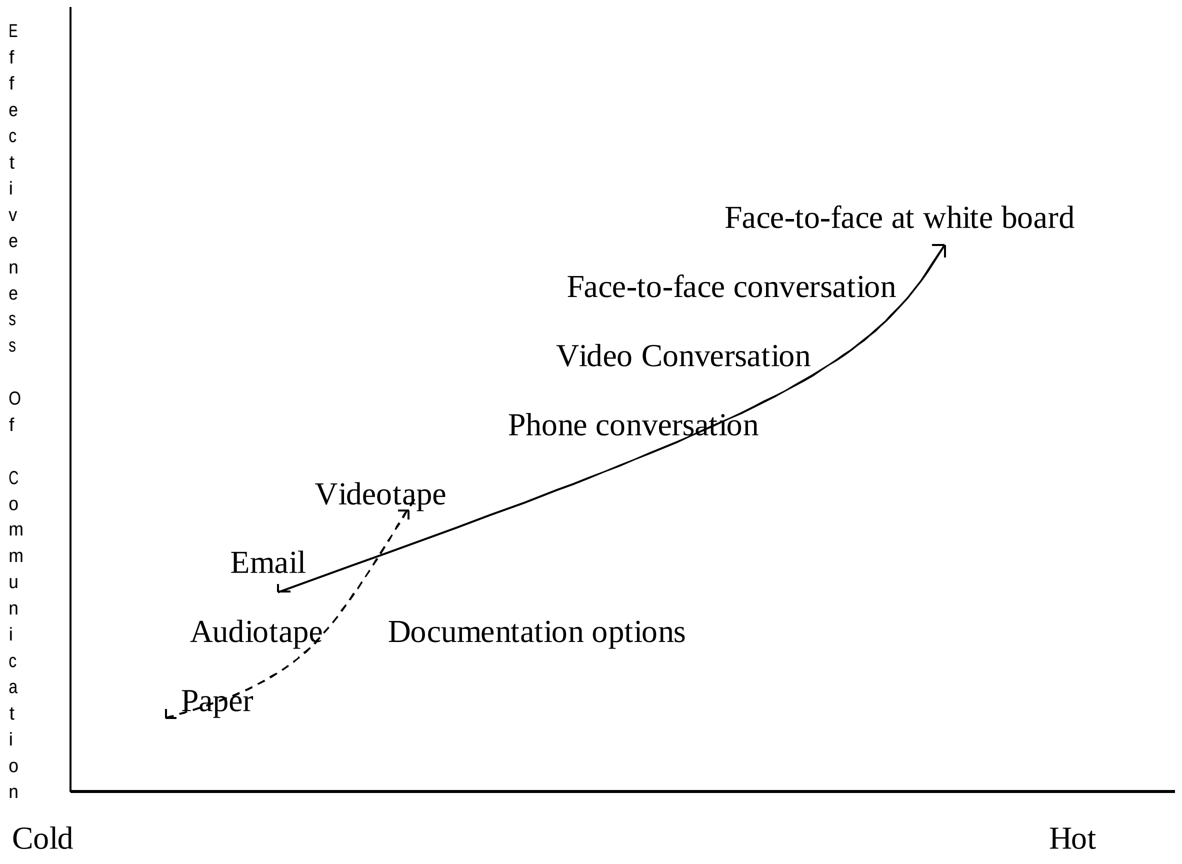

Figure 14: The communication value of different techniques (Korkala et al., 2006)

In Figure 14 Korkala et al. (2006) show the effectiveness of communication in software development at various levels starting from paper communication to face-to-face at white board communication. The effectiveness of the communication is increasing from paper (cold) to verbal and then from verbal to video and then from video to face-to-face conversations (hot).

4.3.3. Continuous integration and Early Testing

One of the key elements in software development is testing. Testing is carried out in software development to check the completeness of the software, whether the software is delivered for what it is developed and to remove the risks associated with the software (Lyndsay, 2007). A set of software engineering practice that increases the speed of the delivery of software by decreasing the integration time is called continuous integration (Stolberg, 2009), but according to

(Bakal et al., 2012) a continuous integration requires every team member to integrate their work regularly on daily basis, and there are multiple integrations every day. Those integrations are verified by an automated build. Automated build is carrying out regression tests to detect integration errors very quickly. This process reduces the integration problems and increases the speed of the software development.

4.3.4. Small release

During the agile development, priorities have been given to the requirements. Some requirements are very important and needs to be developed early. The developers start working on the most important requirements and when they are done, they deliver the functionally developed part of the software to the customer to take benefit from it and give feedback (Aitken et al., 2013). Because of the small release method, the software developed is of high quality, the customer starts the benefit from the software early and about all the requirements are used. The software is on time and within the required budget. Because there is no extra requirements, that can save a lot of development time (Rico, 2013).

According to (Abrahamsson et al., 2002) one of the main issue in agile methods is that agile methods are simple and deliver the software in quick possible time by focusing on the most important functions first, deliver them in quick possible time, and collect the feedback and to react to that feedback.

5. Discussion and conclusions

5.1. Discussion

During this part of the thesis the overall summary of the thesis is given that can cover about all the topics discussed in the entire thesis document and that can attract the attention of the reader**s**. In this section the author is giving a complete clear picture of the software development methods starting from traditional software development methods (SDLC) to the new agile software development methods. The advantages and disadvantages of waterfall software development model, the agile Manifestive and agile software development alliance, different agile methods such as Extreme Programing, Scrum and RUP have been discussed. The difference between the traditional and agile methods has been mentioned. Project success and failure rates are presented and a few agile success factors are considered that can reduce the time and cost of the software development in agile methods.

Software engineering is an engineering discipline that is used for software development. Different companies and organizations need software to run their business and meet their requirements. In early stages like 1970 the software development method used for software development was waterfall software development method. In waterfall software development model the development phases are in sequence, when one phase is complete the output of that phase becomes the input of the next phase, like when the requirement gathering phase is complete then the next phase is system and software design, the requirement gathering phase becomes the input of system and software development. This process is continuous until the software is ready. The problem in waterfall method (traditional method) is that if the requirements are changed then the whole process has to be repeated again and that is much expensive and time consuming and thus the software is late and over budget. Also in waterfall method the testing (unit testing and system testing) is carried out at the last stages of the development if there is any error at that stage then fixing of that error is very difficult, expensive and time consuming and causes delay in the software delivery. Traditional software development team is very large and communication between the team members is very difficult that also results a delay in software delivery and also the software is over budget. There are many disadvantages in traditional software development than advantages. See Table 1.

Due to the high growth of internet and e-Business there is a high competition between the organizations for their business and because of that competition organizations are trying to improve their technology in order to be in the row of competition. For this reason, organizations need software in quick possible time and within the estimated budget. The requirements of organization business are changing continuously because of the changes in their business process. To overcome all those problems and deliver the software product to customer on time and within budget, the software companies required such software development models that are flexible to changes and are able to accommodate any changes at any stage of the software development.

Agile methods are very simple and deliver the software in quick possible time by focusing on the most important functions first deliver to the customer and collect feedback from them and react to that feedback. Moreover in agile methods the requirements and design details are very flexible and can be changed at any stage of the development phase. In agile methods there is less documentation, prototyping is used and the methods are iterative and at each iteration there is continuous integration and early testing (unit testing, integration testing etc.) etc. Agile software is of high quality, on time and within the required budget.

Three agile methods have been studied in this thesis. The process of each method is explained in details, the roles and responsibilities of those methods are mentioned and also where each of these methods can be used is also outlined in section three of this thesis document.

In this thesis the difference between the agile methods and traditions methods have been presented in section 4.1 of this thesis and explained based on different research articles. The success and failure rate of the agile and traditional methods have been discussed in section 4.2, the reasons for the failure of traditional methods have been explained and the reasons of the success of agile methods were also highlighted.

5.2. Conclusions

It’s a common fact that software plays a very important role in business and any type of business are dependent on software. If we go 30 years back then at that time there was not that much competitions in business and the organizations were using their old technology and were fulfilling their business requirements. The global business was not that much as compared to current situations. The software companies were using the traditional software development method like waterfall software development method to develop software product for those organizations. The projects were lengthy (can be up to many years) and there were too many requirements, and even the customer did not know exactly what they required and in majority of cases the projects were not delivered on time and within budget. Traditional software development was not flexible to the requirement changes, there was a communication gap between the team members and customer, and also between the team members, and fixed project plan, fixed software and system design, late testing, not clear cost estimations, and changes were very difficult and expensive and time consuming. Majority of the projects failed or were over budget using the traditional software development methods.

During the current situation due to advancement in the internet field and e-business the business of majority of the companies are expanded globally and the those companies are using very advanced technology due to high competition between the business companies and because of that competition the organizations business requirements are changing at a very fast rate and it is not possible for the software companies to use the traditional software model for their software development. Those companies need such software development methods that are flexible to requirements and changed requirements and can deliver the software product in short possible time and within budget

Agile methods have been used by software companies and the success rate of agile methods is much more than the traditional methods. I argue that agile methods are very simple and easy and can complete the customer requirements easily within time and budget because agile methods involve customer during the software develop phases. There is a close communication between the customer and developers, and also among the developers (team members). In agile methods the software project is broken down to small parts and those parts are developed in different iterations. Agile methods are very flexible to change requirements at any stage in the software development phase, which is the key requirement of today global business. In agile methods the software is developed in small parts and there is a continuous integration and testing in each iteration. Due to the customer involvement in the development phases there are no waste requirements in the development phases, and almost all the requirements are used in the software product. The software developed using agile methods is of high quality, deliver on time, within budget and there is customer satisfaction involved.

Agile methods provide high quality, reduce time and cost in software production because of using its best practices in software development. Those best practices are close customer collaboration, flexible system and software design, accommodate changes at any stage of development, continuous integration and testing, small releases, short project time, short project teams etc.

5.3. Future Studies

The current thesis is about the comparison and problems of traditional and agile methods. What are the problems in traditional software development and how agile methods reduced those problems by using the agile methods best practices. The thesis also focuses on the differences between the traditional methods (waterfall) and agile development.

The same study can be used in future while developing a cost benefit analysis model for software engineering or re-engineering. Input (requirements, new requirements, changed requirements etc.) will be given to the analysis model and the analysis model will give the cost and time estimation for the project and the development team will decide its decision of engineering the product or to re-engineer the product based on the estimations of the analysis model.

Limitations of this thesis are as follows.

The current thesis is limited to the traditional waterfall modal of software engineering and the agile methods. Overall, current thesis work is limited to the topics about the traditional software development and agile software development and there were limited articles about the comparison of waterfall method and agile methods.

References

Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J. (2002) ‘Agile software development methods Review and analysis’, University of Oulu, VTT Publications 478

Ahmad, F., Capretz, L.F. (2008) ‘Best practices of RUP in software product line development’, International Conference on Computer and Communication Engineering, IEEE Publication, PP: 1363-1366

Ambler, S. (2014) ‘Overcoming the Myths of IBM Rational Unified Process (RUP) and Agile Development’ Available at: http://www01.ibm.com/software/info/television/html/M649306B47502P68.html (Last accessed: November 9, 2014)

Anderson, Beattie R., Beck, K. (1998) ‘Chrysler Goes to “Extreme” ’ , A case study, PP: 24-28

Awad, M.A. (2005) ‘A comparison between Agile and Traditional Software Development Methodologies’, the university of Western Australia

Bakal, R.M. , Althouse, J., Verma, P. (2012) ‘Continuous integration in agile development: How agile methods, continuous integration, and test-driven enhance design and development of complex systems’, available at: http://www.ibm.com/developerworks/rational/library/continuous-integration-agile-development/continuous-integration-agile-development-pdf.pdf (Last Accessed: September 20,2014)

Bassil, Y. (2012) ‘A Simulation Model for the Waterfall Software Development Life Cycle’, International Journal of Engineering & Technology (iJET), ISSN: 2049-3444, Vol. 2, No. 5

Beck, K. (1999) ‘Extreme Programming Explained, Embrace Change’, Addison Wesley

Bhalerao, S., Ingle, M. (2010) ‘Analyzing the modes of communication in Agile Practices’, IEEE International Conference on Computer Science and Information Technology, PP: 391-395

Bohem, B. (2002) ‘Get Ready for agile methods: with care’, Vol: 35, IEEE Publication, PP: 64-69

Cao, L., Mohan, K., Peng Xu, Ramesh, B. (2004) ‘How Extreme does Extreme Programming Have to be? Adapting XP Practices to Large-scale Projects’, Proceedings of the 37th Hawaii International Conference on System Sciences, IEEE Publication

Czarnacka-Chrobot, B. (2010) ‘The Economic Importance of Business Software Systems Size management’, Fifth International Conference on Computing in the Global Information Technology, IEEE Publication, PP: 293-299

Devi, V. (2013) ‘Traditional and Agile Methods: An Interpretation’ Available at: https://www.scrumalliance.org/community/articles/2013/january/traditional-and-agile-methods-an-interpretation (last accessed: September 13, 2014)

Denzin, N.K., Lincoln, Y. S. (2011). ‘The SAGE Hankbook of Qualitative Research’, 4th edition, USA: Library of congress cataloging-in-Publication Data. PP 8-9.

Del Maschi, V.F. , Spinola, M.M. , Costa, I.A. , Esteves, A.L. , Vendramel, W. (2007) ‘Practical Experience in Customization of a Software Development Process for Small Companies Based on RUP Processes and MSF’, Portland international Center for Management of Engineering and Technology, IEEE Conference Publication, PP: 2440-2457

Dobb, D. (2010) '2010 It projects success Rates', Available at: http://www.drdobbs.com/architecture-and-design/2010-it-project-success-rates/226500046 (last accessed: September 13, 2014)

Fowler, M., Highsmith, j. (2001) 'The Agile Manifesto' available at: www.drdobbs.com/open-source/the-agile-manifesto/184414755 (Last accessed: April 20, 2014)

Guo, F., Xia, B., Xue, F. (2011) 'Analysis of Software Processes and Enhancement for RUP', IEEE 2nd International Conference on Software engineering and Service Sciences, PP: 295-298

Holmstrom, H., Fitzgerald, B., Agerfalk, P., Conchuir, E. (2006) 'Agile practices reduce distance in global software development. Information system development', Vol 23, Issue 3, PP: 7-18

Hussain, Z., Lechner M., Milchrahm, H. , Shahzad, S., Slany, W. , Umgeher,M. (2008) 'Optimizing Extreme Programming', Proceedings of the International Conference on Computer and Communication Engineering, IEEE Publication, PP: 1052-1056

Hanakawa, N., Okura, K. (2004) 'A project management support tool using communication for agile software development', 11th Asia-Pacific Software Engineering Conference, IEEE Publication, PP: 316-323

Hasnain, E., Hall, T., Shepperd, M. (2013) 'Using Experimental Games to Understand Communication and trust in Agile Software Teams',6th International Workshop on Cooperative and Human Aspects of Software Engineering , IEEE Publication, PP: 117-120

Hayata, T., Han, J. (2011) 'A Hybrid Model for IT Project with Scrum', IEEE international conference on Service Operation, Logistics, and informatics, PP: 285-290

IMB White Paper, (2003) ‘Using the Rational Unified Process for small projects: Expanding Upon eXtreme Programming’, Available at: ftp://ftp.software.ibm.com/software/rational/web/whitepapers/2003/tp183.pdf
(Last Accessed: August 12, 2014)

Juric, Radmila, (2000) ‘Extreme Programming and its Development Practices’, Information Technology Interfaces, proceedings of the 22nd international conference, IEEE publication, PP: 97-104

Jacobson, I. , Booch, G. , Rumbaugh, J. (1999) ‘Unified Software Development Process’ , Addison-Wesley

Kruchten, P. (2000) ‘The Rational Unified Process: An Introduction’, Addison Wesley

Kohlbacher, M., Stelzmann, E., Maierhofer, S. (2011) ‘Do Agile Software Development Practices Increase Customer Satisfaction in System Engineering Projects?’, IEEE international Conference, PP: 168-172

Kähkönen, T. , Abrahamsson, P. (2003) ‘Digging into the Fundamentals of Extreme Programming building the Theoretical Base for Agile Methods’ Proceedings of the 29th EUROMICRO Conference, IEEE Publications, PP: 273-280

Kessel, C. (2013) ‘Software History: Waterfall – The Process That Wasn’t Meant to Be’ available at: http://ig.obsglobal.com/2013/01/software-history-waterfall-the-process-that-wasnt-meant-to-be (Last accessed: March 27, 2014)

Keenan, F. (2004) ‘Agile Process Tailoring and probLem analYsis (APTLY)’, 26th IEEE international Conference on Software Engineering, PP: 45-47

Korkala, M. , Abrahamsson, P. , Kyllonen, P. (2006) ’ A Case Study on the Impact of Customer Communication on Defects in Agile Software Development’, IEEE Agile Conference, PP: 11-88

Kruchten, P. (2002) ‘Tutorial: Introduction to the Rational Unified Process’, Proceedings of the 24th International Conference on Software Engineering, IEEE Publication

Lagerberg, L., Skude, T. (2013) ‘The impact of agile principles and practices on large-scale software development projects’, Linkopings university, SE-581 83 Linkoping, Sweden

Livermore, J.A. (2007) ‘Factors that impact implementing an agile software development methodology’, PP: 82 – 86 IEEE Conference Publications

Lindstorm, L., Jeffries, R. (2004) ‘Extreme programming and agile software development methodologies’, information systems management, Vol 21, Issue 3

Lyndsay, J. (2007) ‘Testing in an agile environment’, Workroom Production Ltd, software testing papers

Manjunath, K. N., Jagadeesh, J., yogeesh, M. (2013) ‘Achieving quality product in a long term software product development in healthcare application using Lean and Agile principles’, conference location Kottayam, IEEE publication, PP: 26-34

Misra, S. C., Kumar, V., Kumar, U. (2009) ‘The success factors of agile software development’ available at: http://ww1.ucmss.com/books/LFS/CSREA2006/SER5088.pdf (Last accessed: December 23, 2013)

Miller, G. G. (2001) ‘The Characteristics of Agile Software Processes’, The 39th international Conference of Object Oriented Languages and Systems, Santa, CA, IEEE publication

Maciaszek, L., Liong B.L. (2005) ‘Practical Software Engineering A Case Study Approach’, England, PP: 5-25

Monteiro, P. , Borges, P. , Machado, R. , Ribeiro, P. (2012) ‘A reduced set of RUP roles to small software development teams’, International Conference on Software and System Process, IEEE Publication, PP: 190-199

Mary Land, ‘Legacy Transformation’, available at: http://doit.maryland.gov/SDLC/Documents/Legacy%20Transformation.pdf (Last accessed: January 04-2014)

Nawrocki, J., Jasinski, M., Walter, B., Wojciechowski, A. (2002) ‘Exreme programming modified: Embrace Requirements Engineering Practices’, IEEE joint international conference on requirement engineering, PP: 303-310

Paetsch, F., Eberlein, A., Maurer, F. (2003) ‘Requirements Engineering and Agile Software Development’, IEEE international workshops on enabling technologies: infrastructure for collaborative Enterprises, Twelfth IEEE International workshop, PP: 308-313

Pressman, R. S. (2001) ‘*Software Engineering’, United* States: Thomas Casson, PP: 19-23

Royce, W. (1970) ‘Managing the Development of Large Software Systems’, Proceedings of IEEE WESCON 26, PP: 1-9

Rising, L., Janoff, N.S. (2000) ‘The scrum software development process for small teams’, IEEE Journals and Magazines, PP: 26-32

Rational Unified Process, (1998) ‘Best Practices for Software Development Teams’, Available At: https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf (Last Accessed: May 8, 2014)

Rico, D. (2013) ‘Agile Methods Cost of Quality: Benefits of Testing Early & Often’, available at: http://www.slideshare.net/davidfrico/rico12z (Last accessed: January 04-2014)

Ruparelia, N.B. (2010) ‘Software Development Lifecycle Models’, available at: dl.acm.org/ft_gateway.cfm?id=1764814&type=pdf (Last accessed: April 23, 2014)

Sulemani, K. A., Nasir, M.N. (2009) ‘Communication Support to Scrum Methodology in Offshore Development’, Available At: hhttp://www.bth.se/fou/cuppsats.nsf/all/cf379a3854ec1378c12576850042c941/$file/Communication%20support%20to%20Scrum%20methodology%20in%20Offshore%20project.pdf (Last Accessed: August 8, 2014)

Sommerville, L. (2011) ‘Software *Engineering’*, United States: Edwards Brothers. PP: 27-36

Salo, O., Abrahamsson, P. (2008) ‘Agile methods in European embedded software development organizations: a survey on the actual use and usefulness of Extreme Programming and Scrum’, Institute of Engineering and Technology, PP: 58-64

Schwaber, K., Beedle, M. (2002) ‘Agile Software Development with Scrum’, Upper Saddle River. NJ, Prentice-Hall

Schwaber, K. (1995) ‘crum Development Process’, OOPSLA’95 workshop on Business Object Design and Implementation. Springer-Verlag.

Schwaber, K., Sutherland, J. (2013) ‘The Definitive Guide to Scrum: The Rules of the Game’, available at: http://www.academia.edu/8370895/The_Scrum_Guide_The_Definitive_Guide_to_Scrum_The_Rules_of_the_Game (Last accessed: April 23, 2014)

Stoica, M., Mircea, M., Gkilic-Micu, B. (2013) ‘Software Development: Agile vs. Traditional’, Informatica Economică, vol. 17 PP: 64-76

Sommerville, L. (2007) ‘Software Engineering’, 8th edition, PP: 6-9

Stolberg, S. (2009) ‘Enabling Agile Testing Through Continuous Integration’, Agile Conference, IEEE Publication, PP: 369-374

Wang Y., Sang, D., Xie, W. (2009) ‘Analysis of Agile Software Development Methods from the View of Informationalization Supply Chain Management’, 3rd International Symposium on Intelligent Information workshop , IEEE Conference Publication, PP: 219-222

Williams, L., Upchurch, R. (2001) ‘Extreme Programming for software engineering education?’, 31st Annual IEEE Frontier in Education Conference, PP: 12-17, Vol. 1

Xiaohua, W., Zhi, W., Ming, Z. (2008) ‘The Relationship between Developers and Customers in Agile Methodology’, International Conference on Computer Science and Information Technology, IEEE Publication, PP: 566-572