Arif Hossain

EVALUATION OF AGILE METHODS AND IMPLEMENTATION

Bachelor's Thesis

CENTRIA UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Information Technology

June 2015

Centria
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Unit	Date	Author
Kokkola - Pietarsaari	June 2015	Arif Hossain

Degree Programme

Information Technology

Name of thesis

Evaluation of Agile Methods and Implementation

Instructor	Pages
Kauko Kolehmainen	54+8

Supervisor

Kauko Kolehmainen

The concepts of agile development were introduced when programmers were experiencing different obstacles in building software in various aspects. The obsolete waterfall model became defective and was no more pure process in terms of developing software. Consequently new other development methods have been introduced to mitigate the defects.

The purpose of this thesis is to study different agile methods and find out the best one for software development. Each important agile method offers own practices, release planning methodology, sprint planning. They differ in sizes and principles. The purpose of this thesis is to attain knowledge about all these facts and understand the agile software development environment. Popular agile methods are analyzed and explained in the theory part of this thesis; where methods have been compared with each other.

The percentage of agile practice is booming. Software development organizations and teams no longer keep faith on traditional development methods. Waterfall method has become obsolete and not effective anymore in building large and complex projects. This thesis presents the reasons and benefits of agile practice and also demonstrates the top software development methods adopted by software development organizations.

At the end of this thesis the guide for Scrum implementation has been presented. Scrum is the most popular lightweight agile method for software development. The Scrum team and the role of Scrum Master have been discussed in detail. The Scrum events and artifacts reveal the key aspects of Scrum implementation.

Keywords

agile, software development, iterative development, agile survey, sprint, scrum team

ABSTRACT

CONTENTS

GRAPHS

ABBREVIATIONS

XP	Extreme Programming
ASD	Adaptive Software Development
DSDM	Dynamic System Development Method
FDD	Feature Driven Development
TDD	Test Driven Development
IID	Iterative and Increment Development

1 INTRODUCTION

The demand of software continues to grow. The use of software and software based products has increased significantly. Modern era is based on technology, and technologies have been applied to make our life easier and flexible. As a result the number of new software building organization has been grown rapidly. New problems and challenges have been find out now a days like defective documentation, process of changing requirements at the middle of project and lack of resources.

The goal of this thesis is to research and study different agile methods and find out the best agile practices through evaluation and comparison. Agile development emphasizes on creating workable software. Traditional software development methods emphasize more on documentation, but agile software development is a new way of building software where documentation is not a big issue. Agile development relies heavily on customer involvement in development process. This thesis presents all key aspects of agile methods and analyses important agile practices which may be effective to select the suitable agile methods for a development team or organization.

Chapter two of this thesis defines the Iterative Software Development. This chapter introduces what Iterative Software Development is and features it contains. It is very important to understand the Iterative Software Development as agile software development is based on the Iterative Software Development where workable software is created iteratively by following small development cycles continuously.

Chapter three contains the main part of this thesis. This chapter covers all key parts of agile methodology and explains the body of agile software engineering. There are various types of agile method. Each method is distinct and offers various practices and flexibilities. All important agile methods are explained with their practices and principles in this chapter. Moreover, this chapter contains comparison, and advantages and disadvantages of key agile methods.

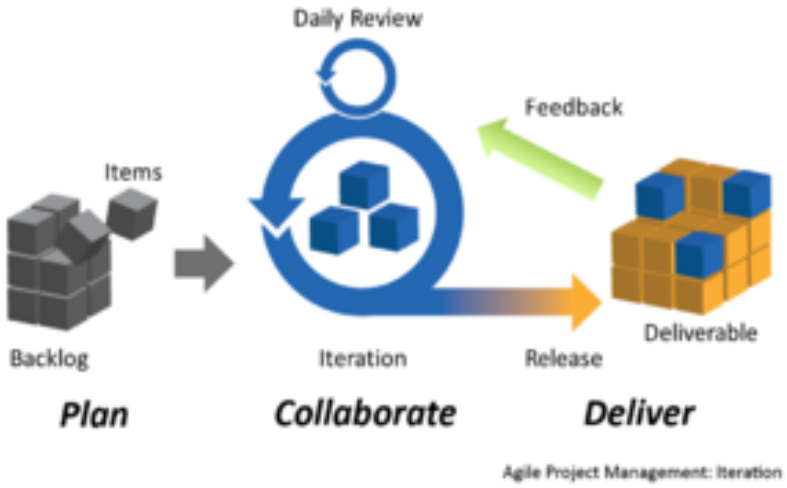
Chapter four is all about agile survey. The survey was conducted in 2014 by the sponsorship of VersionOne. This chapter provides all practical information of agile development; like, how and why a development team or organization adopts agile as their software development method, what are the benefits of agile practice, and what agile methods are mostly used by the development team or organization. Moreover, this chapter presents the main reasons of failure of agile software development. This information is useful indeed to evaluate the best agile practices and apply them in software development process.

Chapter five provides a guideline for Scrum implementation. Scrum is the most popular agile method and its popularity is growing rapidly. This chapter explains the Scrum theory, Scrum team, role of the Product Owner and the Development Team. Scrum Master plays vital role for Scrum success. This chapter provides the services of Scrum Master to the Product Owner, Development team, and organization. The Scrum events and Scrum artifacts are explained in great details in this chapter.

2 ITERATIVE SOFTWARE DEVELOPMENTS

Iterative development approach is to start doing software development in cycles, rather than trying to do everything all at the same time (Wisegeek 2015). “Iterative Development adds agility to the development process” (Wells 1999). Developers divide their development schedule into several iterations of 1 to 3 weeks in lengths. One week seems very short but it is the best choice.

Developer starts with planning stages of the project and moves through each stages in iterative development. On the other hand in obsolete waterfall model software is built in different phases, where each phase follows each other. The problem in this case is there is no way to go back to the previous phases if it needs. The planning phase comes at first of course where all planning and all designs are made before implementation. The testing phase comes when all the implementation work is done. This is very error prone system where error might found late in the project. It prolongs the delivery time as it needs to reschedule to correct the errors and it is tough to change the requirements because all requirements are already listed in the very early phase of the project. However, in iteration development software is developed sequentially with several iterations; where each of the iteration (figure 1) is treated as mini project. (Larman 2004, 10.)



Graph 1: Iterative methodology (Empireone 2010.)

The final outcome of a particular iteration round is an iteration release, a stable, integrated, and tested partially complete system. The iteration releases are released internally not externally. The final iteration release, which is a complete product, is released to the clients or the market. The result of iteration is not a prototype or proof of concept; it is a subset of the final project. (Larman 2004, 10-11.)

In principle the outcome of the each iteration is workable product and independent which can be delivered to the customer. However, in practice, the outcome of the iteration is like a so called demo, workable product which is presented to the customer. Eventually, the customer can get a notion about what has been implemented in the last iteration and what type of product he/she is going to get. Consequently, the customer gets the opportunity to inspect the product and can make new decision to change if it is necessary which can be taken into account in the next iteration phase. (Larman 2004, 13.)

A well-planned schedule is a very important in the every iteration. By managing iteration time boxing the iteration deadline is been determined. Iteration deadline should be taken seriously and it is also important to track the progress during iteration. If it seems that it is not possible to execute the entire task during a particular iteration then it needs to summon an iteration planning meeting to re-estimate the tasks. And remove some tasks which are not important. Attention should goes on the most important tasks as chosen by the customer, instead of having several unfinished works selected by the developers. (Wells 1999.)

3 AGILE

Agile methodology is known as the alternative way to traditional software development process, typically followed in software development. It is a different way of handling software development projects and teams. The use of the word ‘Agile’ in terms of software development derives from ‘agile manifesto’ which had been formed in 2001. A small group of software specialist got together in 2001 to share their opinions and experiences about the traditional software development approaches which were defective and failing far too often. The objective of their discussion was to find out a better option. As a result agile manifesto were introduced, which describes four important values along with 12 broad principles. (Kelly 2007.)

Agile methods are based on a set of values and principles; this is the reason why they are somewhat unique. There was a rapid development in software methodologies in the 1980s and agile methods are the consequences of these trends. Software methodologies like Rapid Application Development and its successor, Dynamic System Development got their maturity with own manifestos, principles, and values. Eventually, agile methods are principles based or a value based software development approach to build new software. (Rico, David, Sayani, Hasan, Sone and Saya 2009, 7.)

3.1 Values of Agile Methods

There are four major values of agile methods, and the values are: Individuals and interactions over processes and tools, collaboration of customer over contract negotiation, working software over comprehensive documentation, and responding to change over following a plan. The success of agile project depends on these values. (Rico et al. 2009, 7 – 8.)

3.1.1 Customer Collaboration over Contract Negotiation

The customers might not have the skills to perfectly specify the system but only they can tell what they want. Though it is hard to work with the customer but it's the reality of jobs. It's important to have a contract with the customer, and having a concept of everyone's responsibilities and rights may form the base of that contract. Successful developers pay especial attention over their customers to find out what their customer's needs and they work closely with their customers. (Ambler 2002, 7.)

3.1.2 Individuals and Interactions over processes and tools

This means programmers are authorized to make their teams and manage themselves. This also means developers are enough competent on computer programming. The fact is; the job of programmer is to build computer software, so they are exceptionally skilled at it. Programmers can get guideline and support from processes and tools make effectiveness better. But all the processes and tools are useless and would not create any results when there is lacking of behavioral and technical skills among the developers. More importantly, this value means also programmers work together to find out a better solution of a complex problems which is not possible to solve by a single person. (Rico et al. 2009, 9.)

3.1.3 Working Software over Comprehensive Documentation

This means building working software is the primary goal of software development, not documentations. It is one of the highest priorities within agile methods. Customers pay for workable software, not for documentation. And this is the way to maximize the business value. (Rico et al. 2009, 9 – 10.)

3.1.4 Responding to Change over Following a Plan

This means being adaptable to alter. Develop software, test it and show it to the customer and change the plan if it is necessary; repeats the process until the customer is satisfied. Customer can change their priorities for different reasons. Moreover, technology changes over the time, there can be change in business environment. These have effects on software engineering. In software development the change is obvious. There should be balance between changing and planning. There should be opportunity to alter the project when situation demands; otherwise the project plan would be irrelevant. (Ambler 2002, 7.)

3.2 The Principles for Agile Software development

The members of the Agile Alliance defined their manifesto into a collection of twelve principles to help people get a sheer understanding of what agile software development is all about. These principles express all key features of agile development. A software development team can achieve knowledge about agile practices from these principles and the team must follows these principles while adopting agile as their software development method. These principles are: Our highest priority is to satisfy the customer through early and continuous delivery of valuable software products. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time scale. Business people and developers must work together daily throughout the project. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. Working software is the primary measure of progress. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely. Continuous attention to technical excellence and good design enhances agility. Simplicity—the art of maximizing the amount of work not done—is essential. The best architectures, requirements, and designs emerge from self-organizing teams. At regular intervals, the team reflects on how to become more effective, and then tunes and adjusts its behaviour accordingly. (Ambler 2002, 7 – 8.)

3.3 Agile Software Engineering

Agile software development harmonizes teamwork, customer collaboration, iterative development, and adaptability to build, operate and maintain new software products. Through flexible and adaptable development process which are quantifiable, systematic and disciplined, the agile software engineering is just-enough, right-sized and just-in-time to get the task done. Agile engineering varies a lot compared with obsolete waterfall model. Time boxed iteration and evolutionary software development which includes planning with continuous changing are the important features of agile engineering. Agile development process includes also evolutionary delivery system, which refers that the software is developed gradually with small portion at a time. Flexible and comfortable manner towards changes is the essential part of agile software engineering. (Rico et al. 2009, 49.)

There is no exact definition of agile methods, and all agile methods contain the same foundation. There are some specific practices which vary in every agile method. Basic practices like short time-boxed iterations with adaptive, evolutionary refinement of plans and goals are been shared by different agile methods. (Larman 2004, 25.)

3.3.1 Requirements

In term of software engineering, software requirements are known as the properties which must be revealed to solve a problem. Moreover, it is also a way of specifying, eliciting analyzing, and validating customer requirements. Software requirements specifications are a major component in traditional methods. (Rico et al. 2009, 50.)

Software requirements are treated as user stories in agile methods, that are simple, very short statements that describe software functions that customer needs to add business

value. There is a process for estimating, writing, sorting, splitting, spiking, and determining the velocity of user stories. User stories can be sorted to prioritize their execution by risk or business value. Customer then can choose the scope of a release by selecting from user stories. The objective of requirements can be attained from user stories. Then user stories can be documented within automated workflow tools or on index cards. Collaboration between developers and customers is the most valuable part of user stories. Within agile methods, user stories are just-enough, right-size, and just-in-time software requirements to maximize business value. (Rico et al. 2009, 50.)

3.3.2 Architecture

Software architecture presents software subsystems and relationship among them. It's also a way of evaluating, specifying, selecting, and identifying an architectural pattern or style. Software architecture description is a vital component in traditional methods. In agile methods, software architecture is considered as system metaphors and it is narratives. It can be used to clarify the functions of the software products and how the products work. There is a way of developing metaphors. One of the objectives of initial iteration is to get a functional skeleton of the system as a whole. Developers build the complete architecture on the basis of user stories that are chosen by the customers. After choosing the user stories, developers and customers combine them into a simple narrative about the total functions and how the full system runs. In term of software engineering body of knowledge, system metaphors present the objective of software architecture. In agile methods, system metaphors are just-enough, right-sized, and just-in-time software architecture to maximize business value. (Rico et al. 2009, 50 – 51.)

3.3.3 Design

Software design is known as the interfaces, components, and characteristics. With the help of software design, it is possible to analyze quality, define software structure, apply design

notations, and utilize software design methods. Software design description is an important component in traditional methods. (Rico et al. 2009, 51.)

In agile development, software design is known as simple design. It is a code design consists of small number of classes and methods which satisfy the user stories. The creation of simple design starts with a unit test for evaluating software methods. When the process of designing and implementation of just enough code is done for getting tests running, the process is iterated; that is, make some code and then remove the useless portion. There must be the reflections of all intentions in the code. Software design is a part of coding in agile methods but in traditional methods it is documentation. (Rico et al. 2009, 51.)

3.3.4 Construction

Software construction means the process of building working software. Moreover, it is also a process of writing code, unit testing, verification, integration testing, and debugging. Agile methods belongs practices for software construction, like pair programming, coding standards, and refactoring, collective ownership. Pair programming means that the two programmers are responsible for coding all software; coding standards refer to the programming language style guides; refactoring refers to the re- structure code whenever needs; and collective ownership refers that every programmers are authorized to alter the programming code whenever needs. (Rico et al. 2009, 51 – 52.)

After getting programming tasks, programmers estimate them, find a suitable partner, make unit tests, maintain style of codding standards, build simple code, and go for the test of the code. Pair programming, coding standards, refactoring, and collective ownership refer the objective of software construction. Software construction is considered as the least important activity in traditional methods, but in case of agile methods, a flexible layer of software developing discipline is held over software construction that makes sure effective software to be built in 14 to 30 day iterations. These types of practices are just-

enough, right-sized, and just-in-time software construction in agile development to maximize business value. (Rico et al. 2009, 52.)

3.3.5 Testing

In software engineering, software testing is the process to evaluate the function and quality of software. Moreover, software testing is the process of developing test plans, procedures, design, script, cases, and reports. Software testing is the major components in traditional methods which document the test levels, progression, conditions, procedures, data, environment and schedules. (Rico et al. 2009, 52.)

There are practices in agile methods for software testing, like continuous integration and test driven development, that are ways of creating unit tests before software coding and integrating all changes into the system baseline to verify them by automated frameworks. In term of software engineering body of knowledge, continuous integration and test driven development provide the objective of software testing. In agile development adaptable process, pair programming, and customer collaboration make a validation and verification life cycle. In agile development, continuous integration and test driven development are just-enough, right-sized, and just-in-time testing for maximizing business value. (Rico et al. 2009, 52 – 53.)

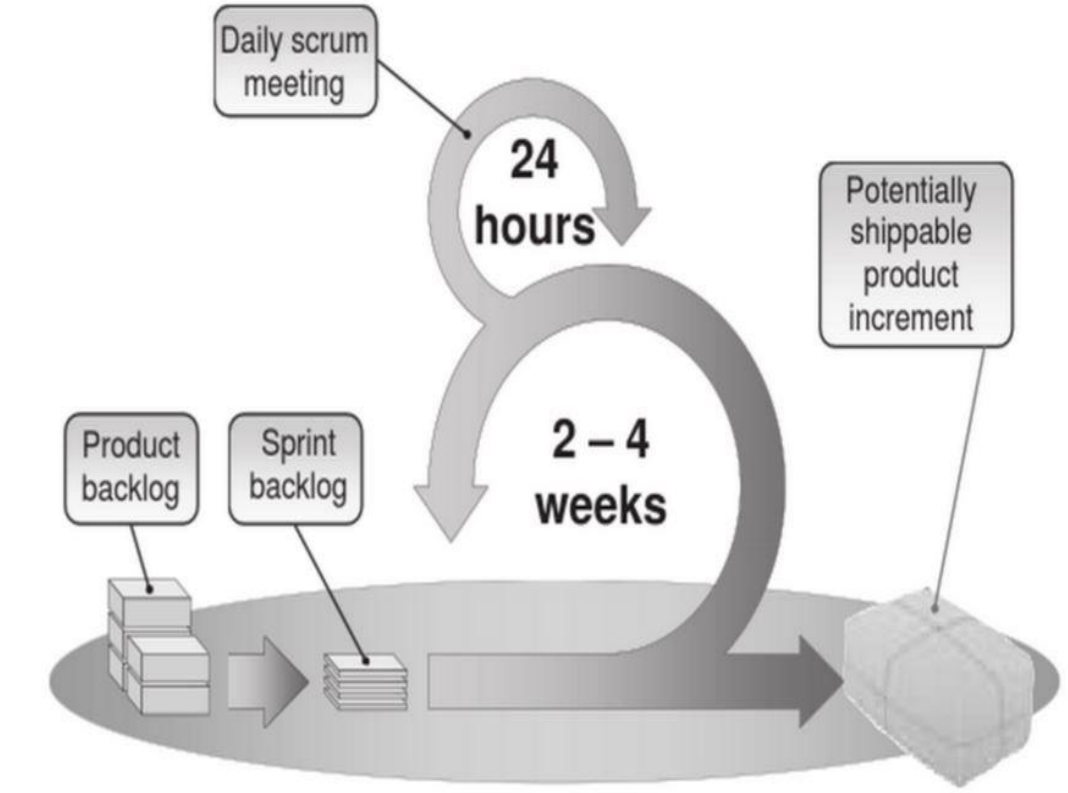
3.4 Types of Agile Methods and Descriptions

There are many types of agile methods. Each agile method has own distinction. They differ from sizes, practices, and flexibilities. Though they differ from each other, the main principles of all agile methods are almost same. All of these methods present the frameworks for developing effective software products. Here are the types and descriptions of most common agile methods in the following. (Rico et al. 2009, 25.)

3.4.1 Scrum

Scrum is the method first created by Jeff Sutherland in 1993, and it is the most common and used agile method in the world. There were two basic reasons for creation of scrum: Existing methods were not so effective, and a new method was necessary to make sure the project success. Scrum considers that there is a need of adaptable software development method because of unpredictable software development and ironclad project plans do not work. Eventually, to make plans successful and help real-world project, Scrum was created. Initially Scrum had three broad phases: Pre-sprint planning, Sprint, and Post-sprint meeting. However, now a days Scrum has five major phases: Sprint planning meeting, Sprint, Daily stand-up meetings, Sprint review meetings, and Sprint retrospective meetings. (Rico et al. 2009, 25 – 26.)

Daily stand-up and retrospective meetings are the significant ideas which are responsible for rich interpersonal communication and process improvement. The use of Scrum is booming in software engineering, where 50% of software engineers prefer Scrum as their software engineering method. (Rico et al. 2009, 26.)



Graph 2: Scrum Process (Rico et al. 2009, 26.)

Two meetings are held before the start of each iteration or Sprint; where in the first meeting, stakeholders refine and re-prioritize the Product Backlog and Release Backlog. They select the goals for the next iteration in this meeting. The Scrum Team and the Product Owner meet in the second meeting, where the objective of this meeting is to determine how to achieve the request, and to make a Sprint Backlog of tasks to fulfil the goals. Another new planning cycle might occur in term of the failure of the estimated effort. Task is formed in 30-calendar iterations; where each is known as a Sprint. (Larman 2004, 117.)

The management and the Scrum Master neither manage the team in achieving iteration goals nor plan the order of tasks. The team is quite free and authorized to find their best process, and find their own way to solve problems.Same special questions are answered by the each team member during a meeting which is held in each workday at the same place and time.Tasks are fixed within a sprint. Management does not add extra works to the team during iteration. Uninterrupted focus is a special feature of Scrum. It is very rare that

something is added, but in this case it is ideal to remove something else. (Larman 2004, 117 – 118.)

It is Scrum Masters duty to ensure that team is not interrupted by external parties with work requests. If it happens then removes them and makes deals with all political and external management issues. The Scrum Master works also to make sure that Scrum is applied, provides resources, removes reported blocks, and makes decisions when he/she is requested. During a meeting, if it reveals that someone is not completing work, and the team is careless about it then the Scrum Master takes proper initiatives to fix this issue. (Larman 2004, 118.)

Blocks reported at the Scrum meeting that need decisions by the Scrum Master are made forthwith, or within one hour. The reported blocks are ideally removed before the next Scrum Meeting. Only the Scrum Team can talk (the pigs) during the Scrum Meeting. Anybody else can attend the meeting if he/she wish but should keep quiet (the chickens) during the meeting. Client-driven adaptive planning is maintained in all iterations. Demo is shown to the external stakeholders at the end of the iteration when the product is workable. The demo can't be shown in terms of ineffective product. (Larman 2004, 118 – 119.)

Scrum is a light framework for the environment; like dynamic and which changes continuously. In Scrum, several variables are taken into account when building software and planning the releases. New requirements must be considered when planning iterations as customer requirements change continuously. In the beginning of the project there are some important things to consider, like time frame, resources, and backup. Each Scrum team should be consisted with less than eight members but multiple teams may build a project and form the increment. Both in small projects and large projects Scrum is used with hundreds of programmers. Scrum teams work in a common project room, where they arrange their daily stand-up meetings. (Larman 2004, 111.)

There is no strict obligation to work in the same working room; it is possible to work in separate room also, but free communication is an essential part of the agile practice

according to agile main principles. Eventually, the open-plan office is the appeasement selection for working space. In case of lacking of free common room, corridor can be used also for daily stand-up meetings. (Larman 2004, 112.)

There are three main phases in Scrum lifecycle; pre-game, development, and post-game/release. Pre-game includes two sub-phases: planning, and architecture. Expectations and vision are set in the planning phase. The initial product backlog which contains all prioritized requirements is made in planning phase. Moreover, in planning phase vision is written, items are estimated, and budget is made. (Larman 2004, 113.)

According to the items in the product backlog list, a high level design of the system is planned in the architecture phase. Initial plans for the content of the releases are done also in the architecture phase. (Coram and Bohner 2005.). The implementation of the system is ready for release in a series of thirty day iteration (Sprints) in development phase. Usually one process consists with around three to eight Sprints before the system is ready for release. Both daily Scrum and Sprint planning meeting are held in all iterations. (Larman 2004, 113.)

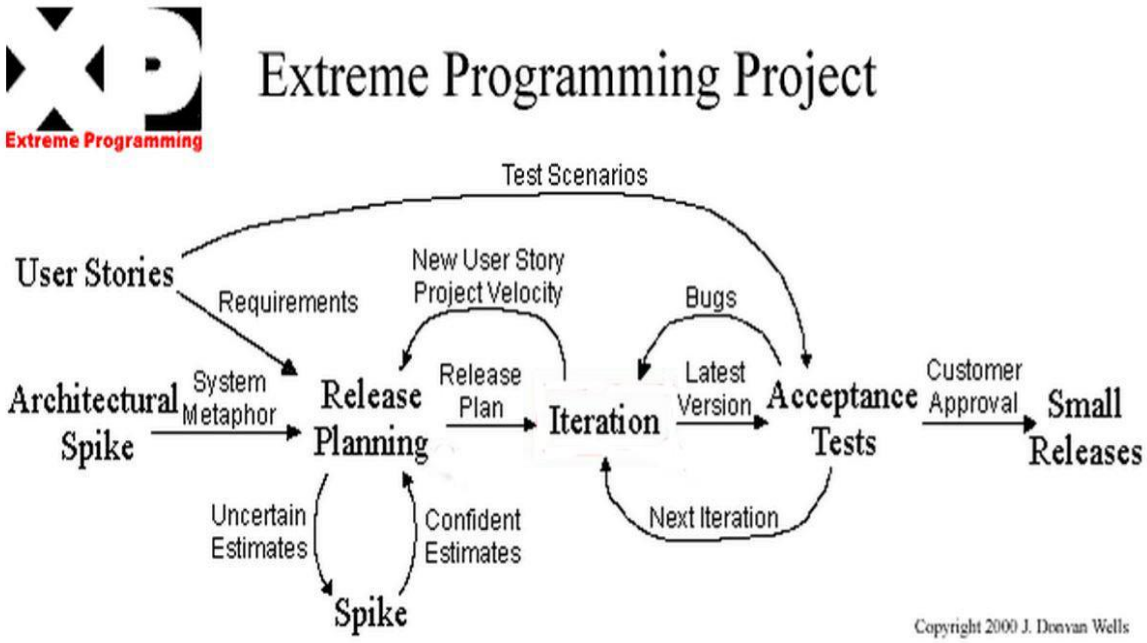
The termination of the release takes place in the post-game phase (release phase). The product backlog list is totally free from requirements and the product is quite ready for release. Release phase belongs marketing & sales, and training also. (Coram and Bohner 2005.)

3.4.2 Extreme Programming (XP)

Extreme Programming was developed by Kent Beck and his team of 20 developers including Ron Jeffries and Martin Fowler at Chrysler in around 1996. It was created to manage a failing payroll system project at Chrysler. Its initial practices were consisted with just-in-time evolution, aggressiveness, self-chosen tasks, communication, and model-driven development. Its key practices extended later into 13 practices that belonged onsite

customers, test-driven development, pair programming, and open workspaces; however, in today, Extreme Programming has over 28 practices and rules for planning, designing, coding, and testing. XP achieved significant public recognition 1999. (Rico et al. 2009, 27.)

Extreme Programming is founded on 4 values; communication, simplicity, feedback, and courage. The main focuses of XP are on skillful development practices, collaboration, and quick and early software creation. (Larman 2004, 137.)



Graph 3: Extreme programming life cycle (Wells 1999)

All the contributors from one team sit together where team must include a business representative (Customer). The responsibilities of the customer are to provide requirements, set the priorities and steer the project. Programmers and testers are the other possible team members. Customer can get the help from analysts to define the requirements. Moreover, there might be a coach and the jobs of the coach are to help the team keep on the track and facilitate the process. The manager manages external communication, coordinates activities, and provides resources. The best teams have no specialists, just only general contributors with special skills. (Jeffries 2011.)

Extreme programming addresses two key questions in software engineering: predicting what will be executed by the due date, and determining what to do next. Planning games refers a set of rules and moves that might be utilized for simplifying the release planning process. It's customer who defines one or more automated acceptance test to demonstrate that the system is working. The team makes the tests and apply them to judge to themselves and to the customer, that the system is executed correctly. (Jeffries 2011.)

The team releases workable and tested software on every iteration. Software is visible, and at the end of iteration, software is given to customer. The customer can use that software for any purpose. Extreme programming teams develop software on the basis of simple design; though it is simple in design but adequate. The XP team makes the design in such a way so that it suits with the system that works here and now. (Jeffries 2011.)

In Extreme Programming, software is developed by two programmers who sit side by side at the same machine. This process makes sure that the code is reviewed by at least one other programmer. This makes the design, testing, and code much better. Every time any programmer releases any code to the repository every programmer tests are run perfectly. Programmer gets quick feedback in this way. In Extreme Programming method, 'Refactoring' is a process which is followed to improve the design continuously. The duplications in coding are avoided and 'cohesion' of the code is enhanced in refactoring process. Refactoring is strongly supported by comprehensive testing to make sure that nothing is broken. (Jeffries 2011.)

Throughout the development, the Extreme Programming teams keep the system quite integrated. The system is never far from a production state in this way. On an Extreme Programming project, any pair of programmers can improve any code at any time. In this way, all the code gets the benefit of many people's attention; as a result, code quality increases and it reduces the possibility of defective code. (Jeffries 2011.)

XP teams maintain a common coding standard system. Eventually all the code in the system looks like that it has been coded by a single individual who is very competent. XP teams form a common vision of how the program works; this is known as ‘metaphor’. In Extreme Programming, team members work hard at a pace that they can go along with for the time being. (Jeffries 2011.)

Extreme Programming is best suited with small and medium sized teams which consist with around three to twenty project members in a team. Communication and coordination between the team members is an essential part of Extreme Programming; hence, wide physical distance among the project members is not allowed. (Larman 2004, 142.)

In the Exploration phase, customer prepares the story cards (features) where they specify the requirements that would be included in the first release. Project team introduce themselves in the Exploration phase, where they present the technology and practices that would be used in project. The prototype of the system can be built also in this phase. The stories that are written by the customer are set into the priority order in the Planning phase. An agreement of the schedule and the content of the first release are made also in this phase. (Larman 2004, 142.)

The Iteration to release phase belongs several iteration before the first release. Each iteration needs one to four weeks to be executed. A system with the architecture of the full system is built in the first iteration. Functional tests that are made by the customer are run at the end of the every iteration. The system is ready for the production after the last iteration. (Coram and Bohner 2005.)

In Productionizing phase, more performance checks and tests are done before the system is ready to release. There might be new list of changes what might still be included in the on-going release. The suggestions and ideas that are postponed are documented for later execution. (Coram and Bohner 2005.)

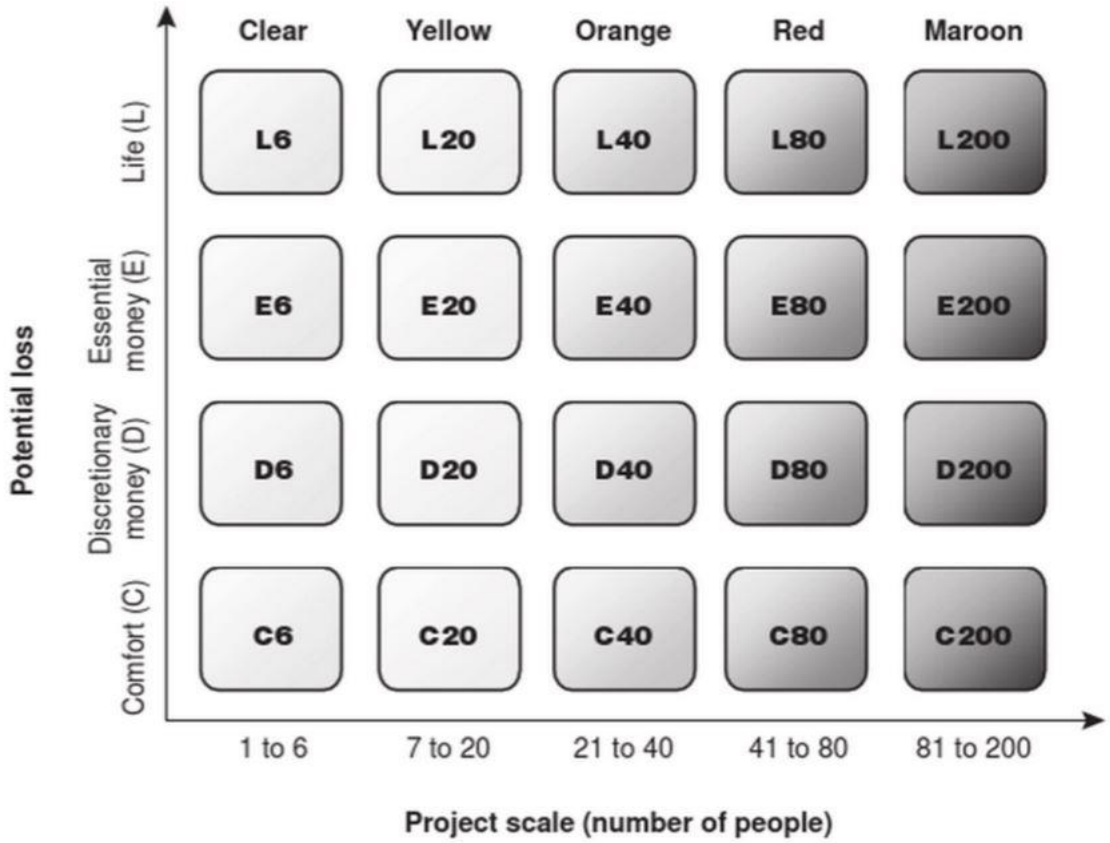
In Extreme Programming, system development and producing new iterations happens simultaneously. Support for the customer is given in the Maintenance phase. The Death phase takes place when all the stories are implemented and the customer does not have any stories to be executed. The documentation of the system is written and there is no more change in system architecture and code. (Coram and Bohner 2005.)

3.4.3 Crystal

The object-oriented programming method Crystal is created by Alistair Cockburn at IBM in 1991. Crystal Method is like a family that consists with 20 distinct agile methods, which are depicted by a two-dimensional grid. Essential memory, imagine life, discretionary money, and comfort on the y-axis and clear, yellow, orange, red, and maroon on the x-axis. There are seven broad stages of Crystal Methods: project cycle, delivery cycle, iteration cycle, integration cycle, week and day, development episode, and reflection about the process. (Rico et al. 2009, 31.)

Crystal Methods is quite a large agile method which consists with five strategies, seven properties, eight roles, nine techniques, and 25 documents. The latest version of this method is a composite of practices from other kind of agile methods. For example, information radiators of Crystal came from Scrum and Extreme Programming, and its burn charts, daily stand-ups, and reflection are from Scrum programming, and its iteration, release plan and side-by-side programming are from Extreme Programming. (Rico et al. 2009, 31-32.)

Crystal method developed with collected software development methodologies, which are communication-centric, people focused, ultra-light and highly tolerant. There are three properties which are central to every Crystal methodology: frequent delivery, reflective improvement, and close communication. (Schuh 2005, 30.)



Graph 4: Crystal Methods (Rico et al. 2009, 32.)

Crystal methodologies have been catalogued on the basis of project size and criticality. Project size is determined by the member of the team and it is marked by the colour; as the darker the colour, the heavier the methodology. Criticality is marked by letter; for example, C: Comfort, D: Discretionary money, E: Essential money, and L: Life. According to the Crystal, as the team becomes larger, a heavier methodology is needed. (Schuh 2005, 31.)

In Crystal methods, incremental development cycles are used and its lengths are not more than four months, however, the recommended lengths are one to three months. There is no definition in Crystal methodologies that which tools, work products, and development practices would be used in the project. Eventually, practices can be adopted from Scrum of Extreme Programming. (Schuh 2005, 33.)

Crystal Orange, Crystal Clear, and Crystal Orange Web are the three main Crystal methodologies that have been constructed. But in practice, only Crystal Orange and Crystal Clear have been constructed and used among these three main methodologies. Crystal Clear is for small project where each team consists with up to eight members who work in the same area. And Crystal Orange is for medium sized projects with 10 to 40 members. (Schuh 2005, 32.)

Policy standards of Crystal Clear and Crystal Orange are, every two to three months, software is delivered incrementally and regularly; progress is tracked by milestones based on software deliveries and key decisions rather than written documents; automated regression testing of application functionality; direct user involvement in the project; two user viewing per release; as upstream is "stable enough to review", downstream activities are ready to start; and at the beginning and middle of each increment, product-and methodology-tuning are held. (Cockburn 2002.)

Within a time frame of two to three months, incremental delivery occurs in Crystal Clear; where in terms of Crystal Orange, the duration of the incremental delivery can be maximum four months. However, in case of delayed delivery, the development teams must negotiate with the customer and avoid less important tasks. (Schuh 2005, 32-34.)

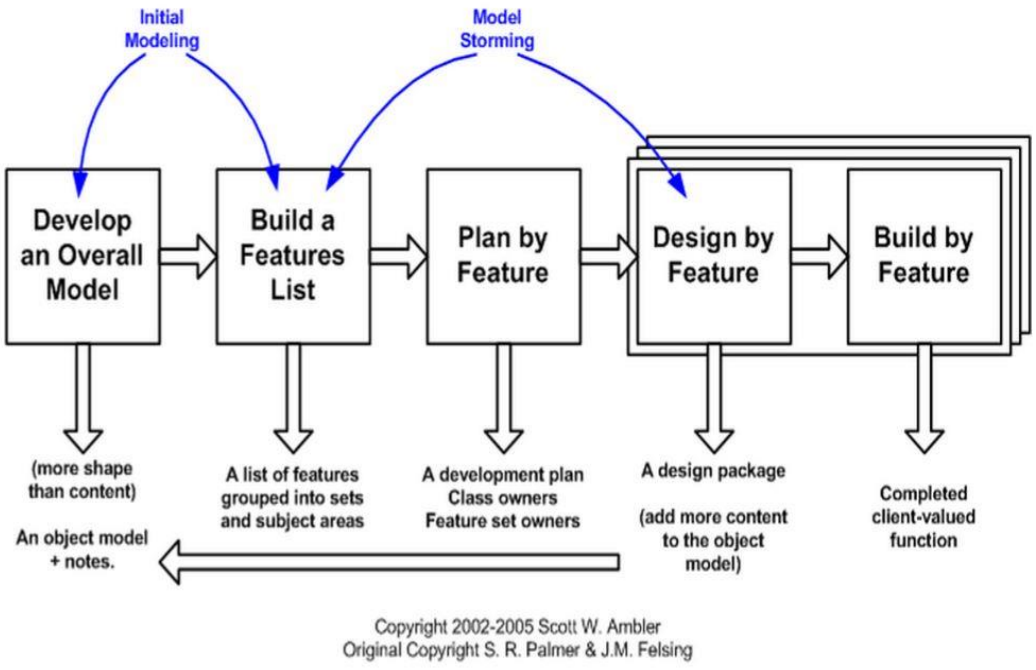
3.4.4 Feature Driven Development (FDD)

Jeff De Luca and Peter Coad developed Feature Driven Development method in 1997 to save a failed banking project in Singapore. Feature Driven Development emphasizes on iterative development, quality and effective software, as well as an adaptable and flexible project management system. FDD consists with five broad phases: develop an overall model, build a features list, making plan by feature, design by feature, and build by feature. (Rico et al. 2009, 30 – 31.)

Feature Driven Development does not focus on entire software development process, its focus is on building phases and design. The agile team can adopt one or more practices from eight main practices of FDD; however by taking all eight practices in use, the best profit can be achieved. (Schuh 2005, 26.)

An overall roadmap to be built for the system. The roadmap is a composed of high-level diagrams that refer the relationships between sequence and class diagrams. Common foundation is maintained to all agile methodologies. Each class is assigned to a particular developer. It is just opposite of Extreme Programming's collective ownership. Due to the involvement of features with more than one class, to design and development in FDD feature teams are the common approach. Attention is paid on the identification of faults. Full system is developed at regular intervals in FDD. Throughout the lifetime of the project; the code, design, analysis, and testing artefacts are to be versioned and stored. Regular and apprehensible updates of status are maintained in FDD. (Schuh 2005, 26.)

Feature Driven Development is consisted with five sequential processes (figure 5). The system is designed and developed completely during these processes. Usually an iteration of a feature needs one to three week period of task for the team. (Ambler 2005.)



Graph 5: The FDD project lifecycle (Ambler 2005.)

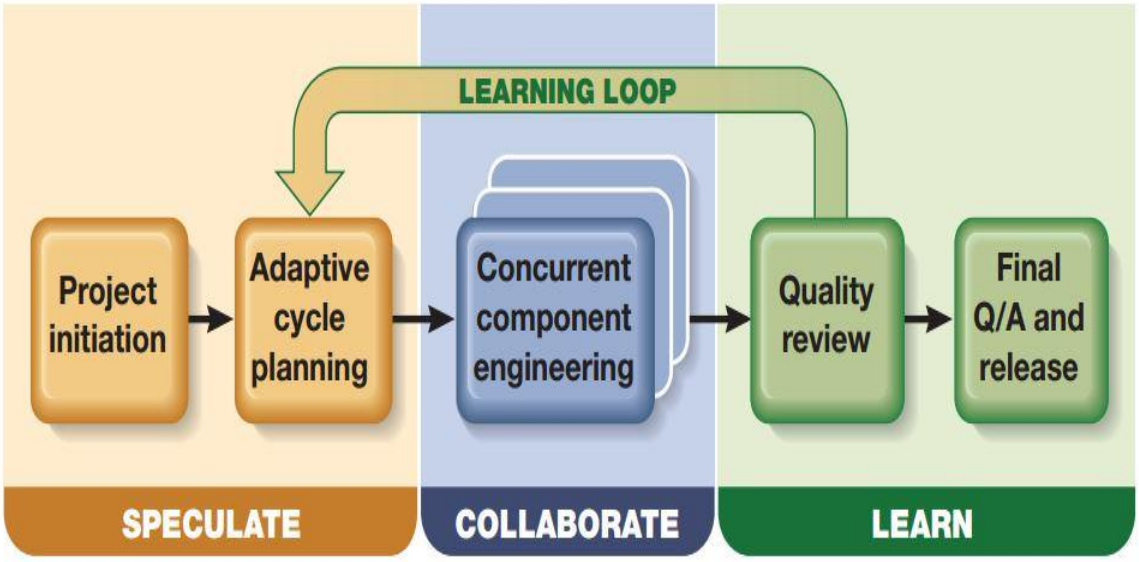
Modelling at FDD is collaborative and time-boxed. Small groups create the domain model in details and then present for peers to review. It is expected that a suggested model or potentially a combination of them would then be used for each area of the domain. Then they will be merged over the time to develop an overall model. (Lawton 2015.)

A list of features is created on the basis of the knowledge that is gained during the modelling process. The list is created by dividing domains into subject fields that hold information on business activities. Some steps are followed for each business activity and these steps represent a categorised list of features. The form of expression of the features is: “action, result, object”. It is hoped that it will not take more than two weeks to complete; but they should be divided into smaller sections if it takes more than two weeks. (Lawton 2015.)

Once the feature list has been created, the following process involves assigning the different feature sets to the developers. Developers the build a design package for each feature with a chief developer selecting a group of features that should be built within a two week period. While refining the model, the chief developer would create diagrams in details for each feature. After completing this, prologues are developed and a design inspection is carried out. Once the design inspections are done, designers make a plan of activities for each feature and create the code for their respective classes. The feature is pushed to the main build when the inspection is made and a unit test carried out. (Lawton 2015.)

3.4.5 Adaptive Software Development (ASD)

Adaptive Software Development is focused at the project management level and its relationship with the organization it serves. Developers encounter problems in building large and complex systems; ASD emphasizes on those problems. In Adaptive Software Development, a dynamic Speculate-Collaborate-Learn lifecycle replaces the static Plan-Design-Build lifecycle. ASD project is handled with in cycle. There are three phases in each cycle (Figure 6): speculate, collaborate, and learn. (Schuh 2005, 36.)



Graph 6: Adaptive Software Development lifecycle (Highsmith 2000.)

Speculate phase includes seven steps: conduct the project initial phase, determine the project time-box, determine the optimal number of cycles and the time-box for each cycle, create an objective statement for each cycle, allot primary components to cycles, assign technology and support components to cycles, and develop a project task list. (Highsmith 2000, 26.)

Delivering the working component is the purpose of Collaborate phase. In Adaptive Software development the main focus goes on the collaboration across the project team. For adopting collaboration within a project, there is no particular procedure recommendation from ASD. It is possible to adopt collaboration practices from Extreme Programming method. For example, for small teams; collective code ownership and pair programming are the suitable practices. (Highsmith 2000, 27.)

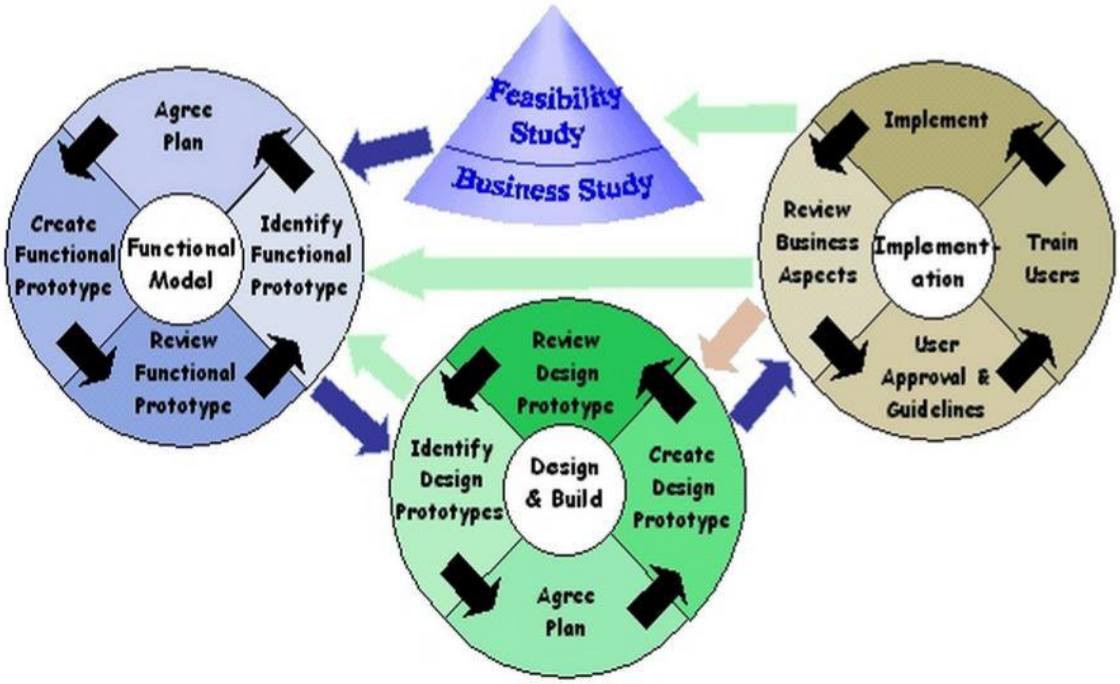
The main purpose of the Learn phase is feedback giving. There must have a review after ending of each iteration. The facts that take place in the review are: delivery teams' functions and practices utilized by the team, project status, and the quality of result of customers and technical point of view. (Highsmit 2000, 27.)

Adaptive Software Development life cycle has six characteristics. Mission focused: all the functions and tasks in every development cycle must be according to the ultimate project mission. Featured based: building activities are not set to task-oriented. Focus goes to building effective software by developing the system a small piece at a time. Iterative: according to the feedback from customers, components are developed over several iterative cycles. Time-boxed: project teams set their activities and functions according to the deadline of release. Project teams work hard to maintain the time frame. Risk driven: high-risk items should be developed as soon as possible. Change tolerant: programmers would constantly assess whether the items they have built are probably to alter. (Orr 2001, 175.)

3.4.6 Dynamic System Development Method (DSDM)

Dynamic System Development Method was created in 1993 by a British consortium of 16 academic and industry organizations. In UK, it is very popular for Rapid Application Development approach. At the beginning DSDM was consisted with three critical success factors: communication between end users and programmers, highly efficient programmers, and flexible customer requirements. Dynamic System Development method focuses on product versus process, integrated configuration management, fulfilling top-priority customer needs, and testing. (Rico et al. 2009, 27.)

In feasibility study stage of DSDM, it is determined that if Dynamic System Development method is suitable for the project. Technical feasibility of the project is evaluated also. Business process flows are assessed in business study stage. Development of functional models of the components takes place in functional model iteration stage. Prototypes are produced iteratively until quality products can be developed. The prototype is fleshed out and tested in system design and build iteration stage. In implementation stage, prototypes are applied to development and transferred into production. (Rico et al. 2009, 27 – 30.)



Graph 7: The DSDM Development Process (Clifton and Dunlap 2003.)

The people for whom the product is going to build must be involved with the development process. This is an important fact which leads the team to develop effective products. In DSDM, project teams are authorized to take instant decisions to make the project successful. Dynamic System Development Method focuses on frequent release. At the crucial stages of the product development, frequent releases allow for user input. They make sure that the product is ready to be released quickly at all times. (Clifton and Dunlap 2003.)

One of the important keys to DSDM's success is incremental product development process. It helps to break the bigger task into smaller task which makes the complex task easy. The most prioritized features and functionalities are developed as early as possible. It might need to change the project concerning issues at any stage of the project. The ultimate project aspects might not match with the project aspects it was before. Hence Dynamic System Development Method emphasizes on having opportunity for change in development cycle. (Clifton and Dunlap 2003.)

High level initial requirements are been executed in the initial stage of the project. Testing is made after each iteration that ensures the working product. Collaboration and team cooperation is the main strength of DSDM. It is essential for the success of the project. Not only the project team, but all involved parties must cooperate to achieve the business objective. (Clifton and Dunlap 2003.)

3.5 Comparison of Agile Methods

Method	Collaboration	Teamwork	Working software	Adaptability
Crystal	• Interaction design	• Daily standups	• Frequent delivery	• Blitz planning
Scrum	• Sprint reviews	• Daily standups	• 30-day sprints	• Sprint planning
DSDM	• User involvement	• Daily meetings	• Iterations	• Feasibility study
FDD	• Domain walkthrough	• Feature teams	• Feature build	• Feature planning
XP	• Onsite customers	• Pair programming	• 14-day iterations	• Release planning

Graph 8: Comparison of practices of different agile methods (Rico et al. 2009, 68.)

It is very important to choose an ideal agile method for a particular project. There are many lower-level practices in all agile methods practice lists. Agile methods belong some traditional practices like, risk management, inspections, configuration management, object-oriented design, prototyping, retrospectives. Programmers like to combine various agile practices from different agile methods and apply them in their project; because it is extremely hard to get all necessary practices from any particular agile method. The complexity, project length, resources, susceptibility to risk, stability of the requirements give variation to the projects. So if it needs to adopt some other agile methods in the middle of the project then it could create confusing atmosphere in the team. Hence, project teams must choose the perfect one or combine the best practices from different agile methods before starting a project. The following graphs demonstrate a short comparison of practices of different agile methods (Graph 8), and the pros and cons of different agile methods (Graph 9) (Rico et al. 2009, 67 – 69.):

Method	Pro	Con
Crystal	• Scalable family of practices • Collection of agile methods practices • Support for code reviews and inspections	• Amalgamation of agile methods practices • Traditional requirements methodology • Resembles traditional methodology
Scrum	• Adaptable planning framework • Adaptable requirements model • Emphasis on self-organizing teams	• No code reviews or inspections • No comprehensive testing methodology • Few customer interactions and automation
DSDM	• Strong customer involvement • Use of empowered project teams • Some support for iterative development	• Spinoff of Rapid Application Development • Resembles traditional methodologies • Weak iterative development model
FDD	• Semi-adaptable planning framework • Support for iterative development cycles • Support for code reviews and inspections	• Traditional requirements model • Significant up-front architecture and design • Few customer interactions and automation
XP	• Adaptable planning framework • Adaptable requirements model • Comprehensive testing methodology	• No code reviews or inspections • No support for virtual distributed teams • Viewed as a set of rules rather than tools

Graph 9: Pros and cons of different agile methods (Rico et al. 2009, 69):

3.5.1 Advantages and Disadvantages of Different Agile Methods

Scrum is a light process framework; it is possible to combine this method with process and practices from other detailed defined methods like Extreme Programming. The core of this method is self-organized team and teams strive hard to achieve the goals. There is no programming practices definition of Scrum for the implementation phase, but Scrum defines the expectations from the team and basic practices for the team in order to attain the goals. One of the important practices of Scrum is 15 minute daily start-up meeting, where team members answer the same accurately defined questions. It draws good impact on the team work and self-organization raises the motivation level. (Rico et al. 2009, 69 – 73.)

In Scrum all alteration are denied after the sprint requirements have been explained at the initial period of each sprint. It is good for the team as it ensures some time to execute all

the works needed to finish the sprint requirement list. Sprint of one month may cause unacceptable delay, if customer is demanding a prompt solution for specific problem. Actual worst case delay may be then close to two months before the solution is seen in release. Scrum splits the problem into small pieces that are easily manageable for the team. (Rico et al. 2009, 69 – 73.)

One of the major advantage of Extreme Programming is it is enriched with wide range of information sources and it is widely used agile method. The actual management and programming practices are more detailed defined in Extreme Programming than to Scrum. Moreover, the process of iterations and frequent builds are well-arranged in Extreme Programming. The negative site of XP is on-site customer requirement is not possible to fulfil as Extreme Programming is applicable for small teams that are consisted with 5 – 10 programmers only. (Rico et al. 2009, 69 – 73.)

In Extreme Programming, developers are allowed to introduce new requirements and issues on the fly, which is effective in some situations to take quick steps to customer needs. The number of format meetings is reduced, and it is an effective way to share information through daily stand-up meetings. Extreme Programming is somewhat free-formed. (Rico et al. 2009, 69 – 73.)

Through reducing the amount of documentation, Extreme Programming increases production efficiency. It could be regarded either good or bad thing. In this way the product may be ready quickly but the product would be delivered to customer without updated documentation. Extreme Programming provides many use full practices but most often it is not possible to get a complete solution for a project through that practices. (Rico et al. 2009, 69 – 73.)

Crystal does not define any particular practices and tools for the project, but those could be adopted from other methods. Crystal has various type of process for small and large project, as a result finding suitable methodology is easy in here. The type Crystal clear is for small group consisted with maximum six programmers, and it needs a shared office

space. However, for a medium-sized project and project team Crystal Orange is suitable. Crystal Orange defines effective communication between the team members. Each team working in the project should have a test engineer as testing is considered as integral part of the development. However, Crystal is not widely used method and it resembles with traditional methodology. And it is the amalgamation of different agile practices. (Rico et al. 2009, 69 – 73.)

Feature Driven Development defines model centric design which could create some impact on new starting team, and especially continuing old projects with no existing models. Individual code ownership is the weakest practice of Feature Driven Development. It increases the risk level in schedule wise, for instances if some key developers get sick in critical development phase. Unlike other agile methods, iteration is not tightly defined in Feature Driven Development. Eventually, some more tasks to be done in start-up phase for the process definition. (Rico et al. 2009, 69 – 73.)

Feature Driven Development is scalable for various types of projects; it defines practices and process for big teams also, and shows how to work multiple teams in parallel. But the iteration content is not as well defined as other agile methods do. In Feature Driven Development, testing is the mandatory part for the process and inspection is applied to remove defects and enhance the product quality. (Rico et al. 2009, 69 – 73.)

Customer feedback and iterative development component-by-component are addresses in Adaptive Software Development collaboration. There is a well-defined general guideline for development process in ASD. Through the general guidelines the project team can analyze and tailor the best suited practices which fit with the team requirements. On the other hand, the disadvantage is, the practices that are described in Adaptive Software Development are strict. So, the project team needs more work and time to apply the practices in the project. (Rico et al. 2009, 69 – 73.)

Among all agile methods, Dynamic System Development method is considered as a most formal agile method. In the development process, there is much architectural design in the beginning of the projects. Moreover, project team needs more documentation as well.

Testing is prioritized and each project team needs at least single test engineer that is a good sign for quality product. However, disadvantages of Dynamic System Development are, the described process seems very heavy, and the access of materials that are described in the practices are charged and controlled by Consortium. (Rico et al. 2009, 69 – 73.)

4 AGILE SURVEYS

This chapter is all about agile surveys. The survey was conducted between July and October in 2014 by the sponsorship of VersionOne. The survey invited individuals from a broad range of industries in the global software development community. The practice of agile is growing day by day. According to the survey in 2014, it is obvious that companies are embracing agile to deliver software faster, easier, and smarter. Among the surveyed organizations, 94% organizations now practice agile. Most of the respondents (87%) marked 'Ability to manage changing priorities' as the top benefit of agile, and 'Team productivity' stays in second position (84%) where 'Project visibility' took third position (82%). (Versionone 2015, 2.)

Around 53% of all respondents had more than 1000 people in their software organization, 35% had more than 5000 people in their entire organization, and 20% worked in very large organizations with more than 20000 people. Most of the questions were answered by Project Manager and Development Staff, total 46%. (Versionone 2015, 4.)

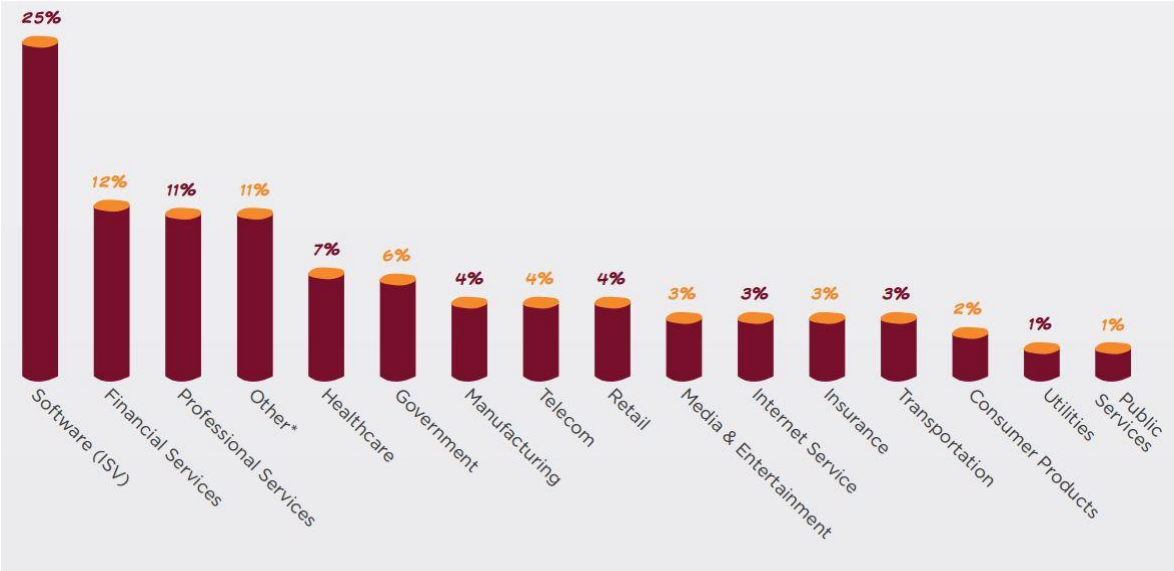
Graph 10: The percentages of respondents (Versionone 2015, 4.)

Total 86% of respondents were from North America and Europe, and 14% respondents were from rest of the world (see Graph 11). Most of the respondents are from software/ISV companies (25%). Moreover, a big number of respondents worked in financial service

(12%) and professional services (11%) organizations (see Graph 12). (Versionone 2015, 4 – 5.)

Graph 11: Geo-location of respondents (Versionone 2015, 4.)

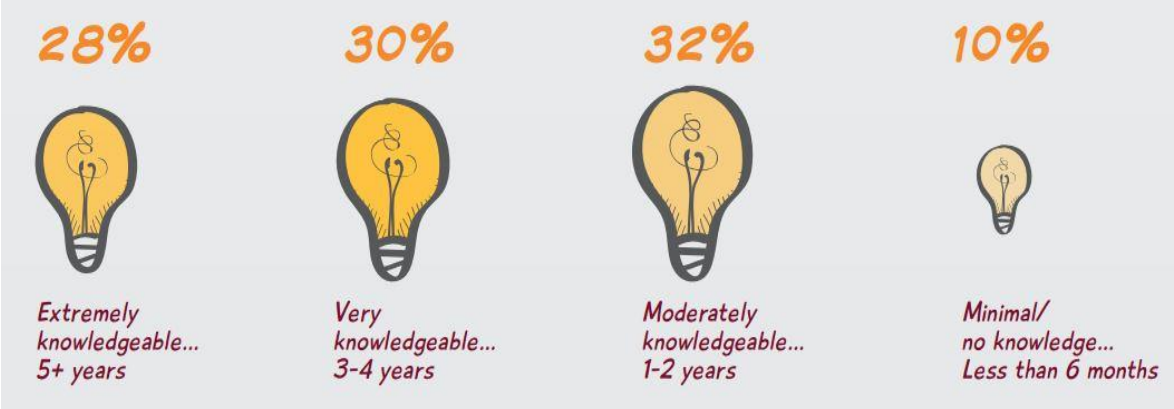
The Survey was not conducted only in software development organization. The graph (Graph 12) indicates that there were various types of organizations in where the survey was conducted.



Graph 12: The percentages of industries (Versionone 2015, 5.)

Among all respondents around 90% said they are at least knowledgeable about agile techniques. Moreover, 58% said they are ‘very’ to ‘extremely’ knowledgeable about agile

techniques (see Graph 13). Among the surveyed organizations (in 2014) 94% of organization practice agile and 24% of respondents worked in companies that have followed agile techniques for greater than 5 years; where it was 19% in 2013 (see Graph 14). (Versionone 2015, 5 – 6.)



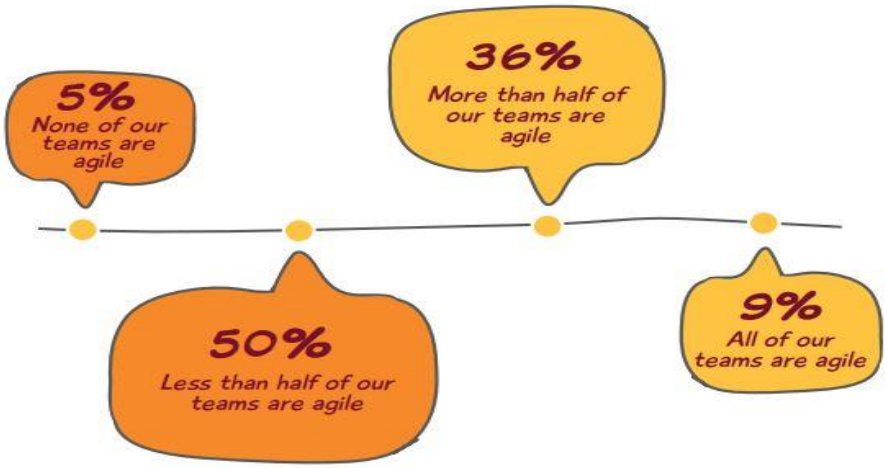
Graph 13: Personal experience about agile techniques (Versionone 2015, 5.)

It is clear that the organizations which are practicing agile as their software development method, they have significant number of software development specialists who can handle agile techniques quite well.

	2014	2013
<1 year:	15%	8%
1-2 years:	29%	40%
3-5 years:	32%	33%
5+ years:	24%	19%

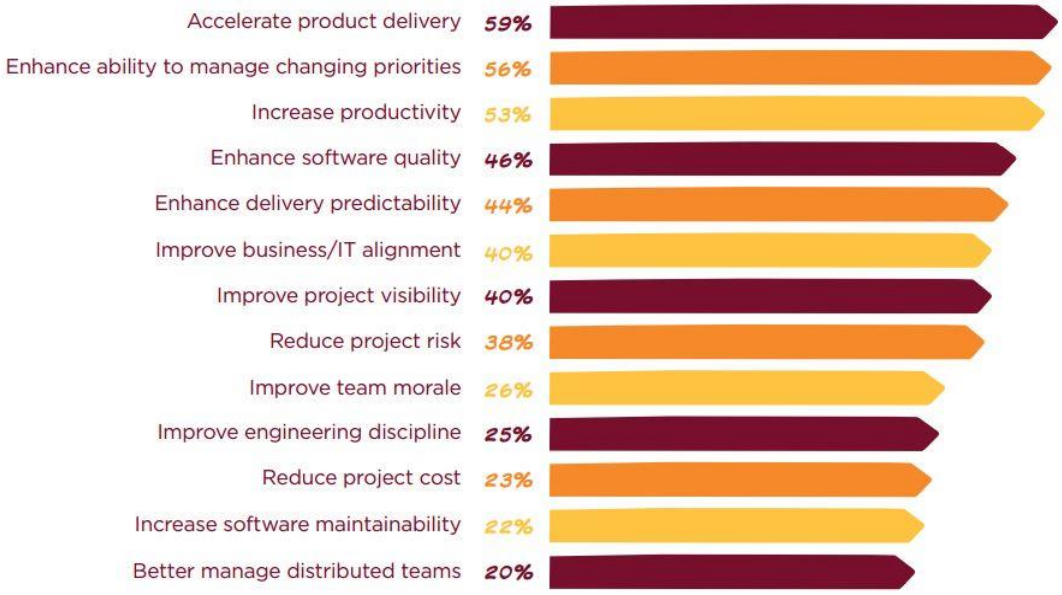
Graph 14: Company experience (Versionone 2015, 6.)

A total of 45% of respondents said they work in such development organizations where most of their teams are agile and only 5% respondent work in non-agile organization (see Graph 15). The top rated reasons for adopting agile are: accelerate product delivery (59%), enhance ability to manage changing priorities (56%), increase productivity (53%), enhance software quality (46%), and enhance delivery predictability (44%) (see Graph 16). (Versionone 2015, 6 – 7.)



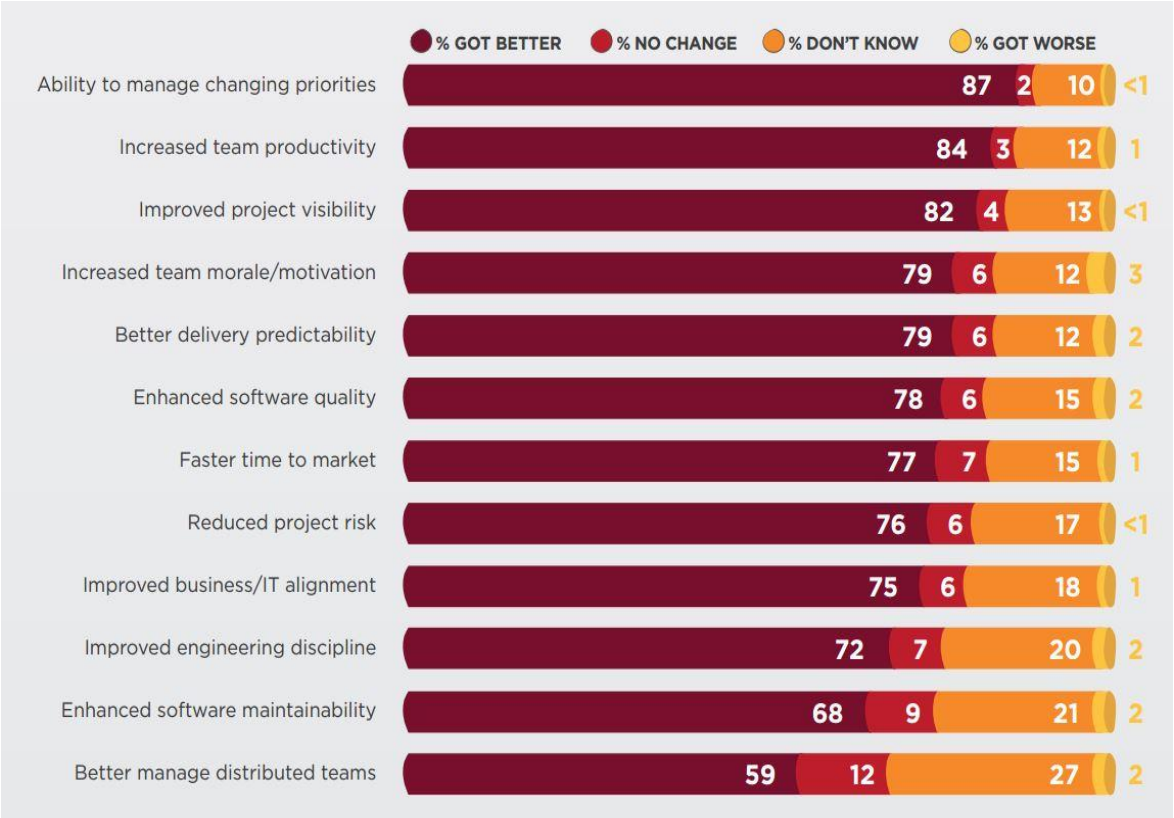
Graph 15: Percentage of teams using agile (Versionone 2015, 6.)

Agile techniques make the software development process fast and flexible. The traditional development methods are not effective enough and it is very time consuming function. Eventually, developers prefer agile techniques so the development process runs fast.



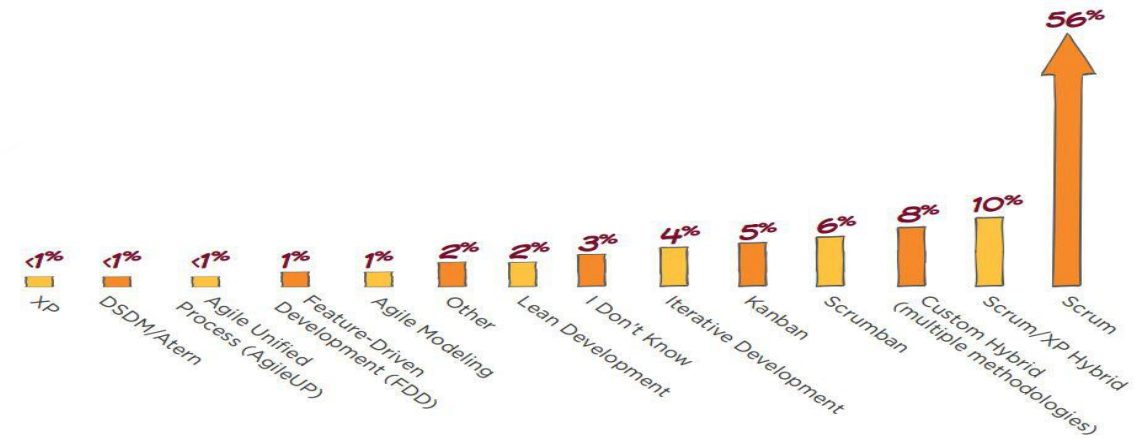
Graph 16: Reasons for adopting agile (Versionone 2015, 7.)

The top three benefits of adopting agile remain same for last four years. They are, manage changing priorities (87%), team productivity (84%), and product visibility (82%) (see Graph 17). However, there is extreme dominance of Scrum over other agile methodologies; 56% of respondents marked Scrum as their used methodology (see Graph 18). (Versionone 2015, 8 - 9.)



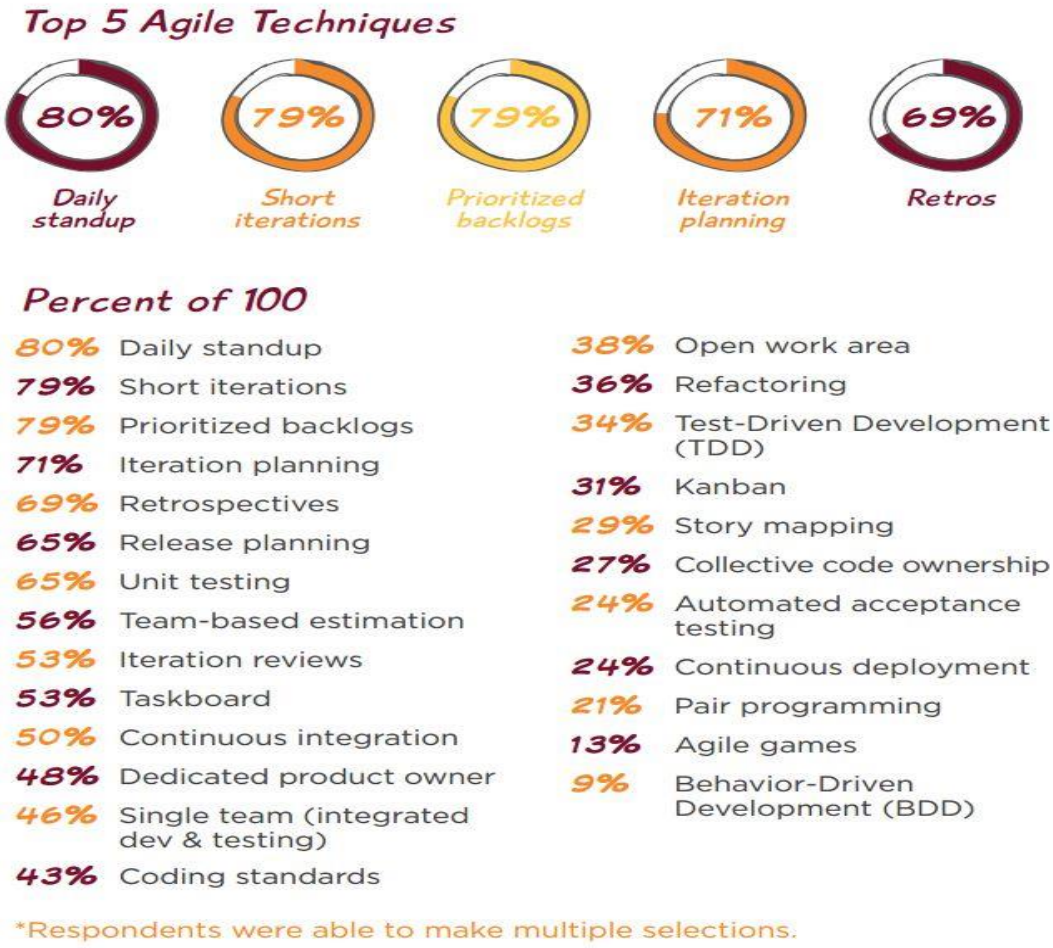
Graph 17: Benefits of agile (Versionone 2015, 8.)

The Graph 17 demonstrates the effectiveness of agile practice. The condition of the organizations got better after adopting agile practices, and the Graph 18 presents the popularity of Scrum. The combination of practices from Scrum and Extreme Programming (XP) is popular also.



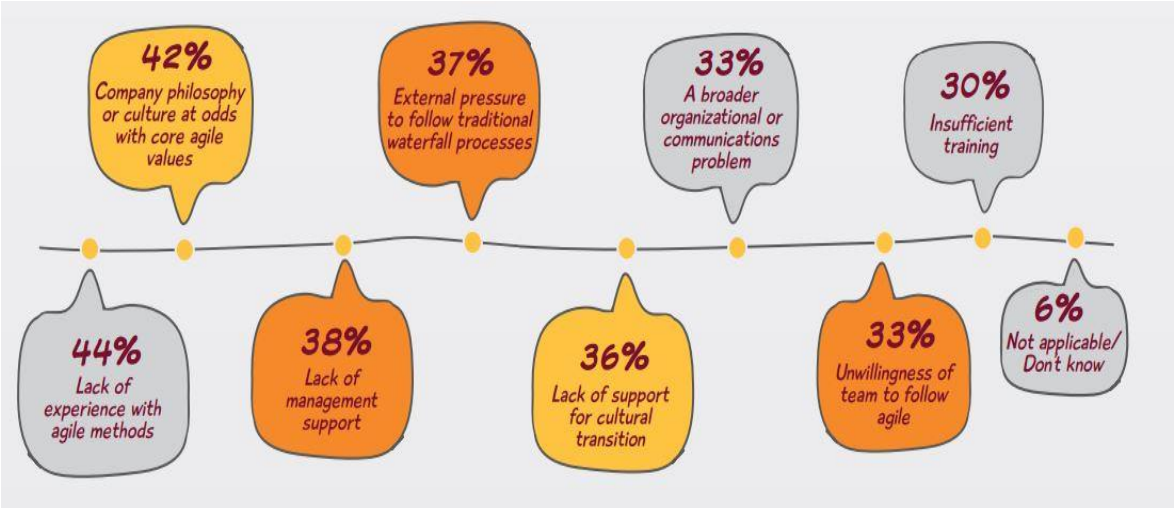
Graph 18: Used agile methodologies (Versionone 2015, 9.)

The most practiced agile technique is the daily stand-up (80%), followed closely by the use of short iterations (79%) and prioritized backlogs (79%) (see Graph 19). In cases of unsuccessful agile project, the top five reasons are (respondents were able to make multiple selections): lack of experience with agile methods (44%), company philosophy or culture at odds with core agile values (42%), lack of management support (38%), external pressure to follow traditional waterfall processes (37%), and lack of support for cultural transition (36%) (see Graph 19). (Versionone 2015, 9 – 10.)



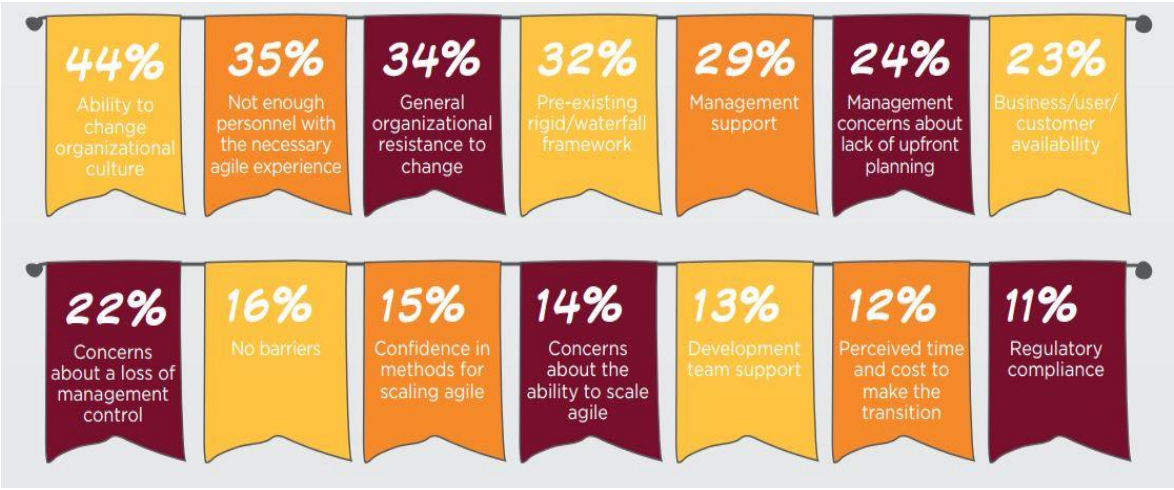
Graph 19: Used agile practices (Versionone 2015, 9.)

In agile practice, the development team arranges a daily short meeting to synchronize the tasks and inspect the project functions so the project goal could be achieved before the product delivery deadline. It is the top rated practice of agile development.



Graph 20: Reasons of failed agile projects (Versionone 2015, 10.)

Respondents marked (respondents were able to mark multiple selections) organizational culture or a general resistance to change as their biggest barriers to further agile adoption (see Graph 21). (Versionone 2015, 10.)

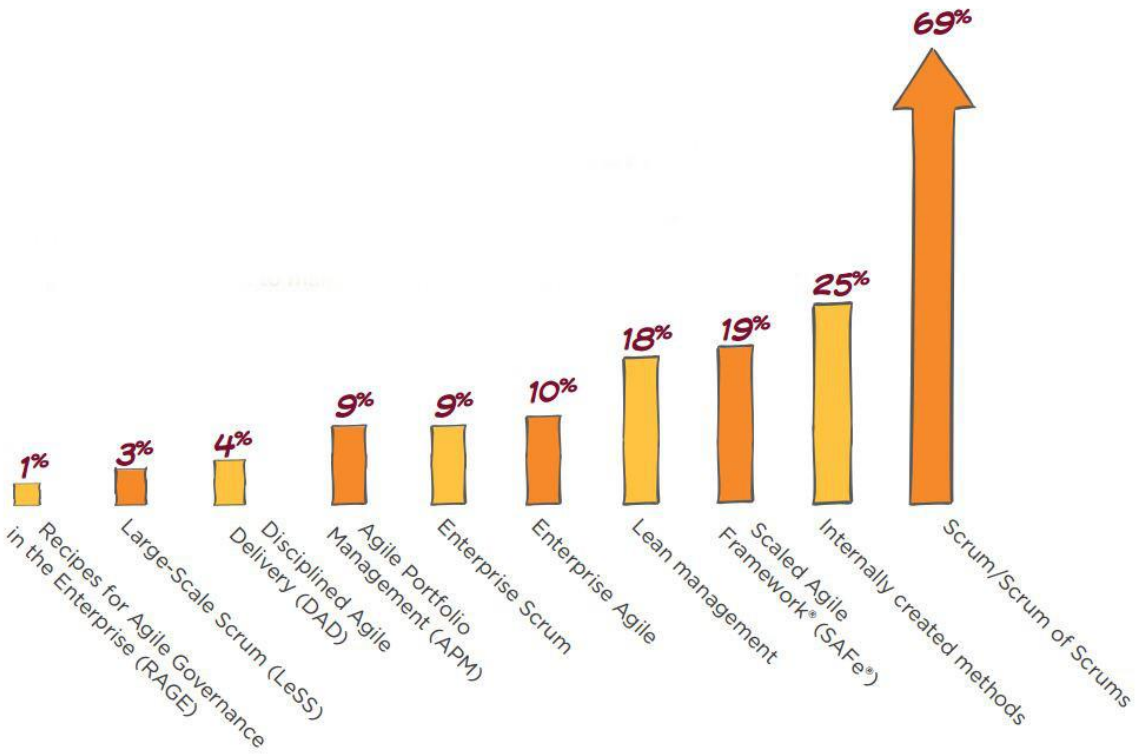


Graph 21: Barriers to further agile adoption (Versionone 2015, 10.)

When asked how respondents gauge the success of their agile initiatives, the most cited value indicator was on-time delivery of projects followed by product quality and customer satisfaction (respondents were able to mark multiple selections) (see Graph 22). (Versionone 2015, 11.)

Graph 22: The way of success measuring (Versionone 2015, 11.)

The majority of respondents use Scrum/Scrum of Scrums to help scale agile within their organizations (respondents were able to mark multiple selections) (see Graph 23). When asked what has been the most valuable lessons learned in easing their adoption at scale, respondents cited these as the top five tips: consistent process and practices (42%), executive sponsorship (40%), implementation of a common tool across teams (39%), agile consultants or trainers (35%), and internal agile support team (31%). (Versionone 2015, 13.)



Graph 23: Scaling methods & approaches (Versionone 2015, 13.)

5 SCRUM IMPLEMENTATION GUIDE

Scrum is a popular framework where complex products are being developed and sustained. It is a lightweight framework and easy to understand. The definition of Scrum can be addressed in this way, “A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value”. (Schwaber, Kubacki, and Sutherland 2012, 136.)

Within Scrum’s framework it is possible to employ various techniques and process. This process framework has been used to handle sophisticated product development since the early 1990s. Scrum ensures the relative efficiency of product management and development practices of developers so that the developers can improve. The Scrum framework is consisted with Scrum Team and their associated roles, rules, events, and artefacts. Within the Scrum framework each component serves a particular purpose and they are very important to Scrum’s usage and success. (Schwaber et al. 2012, 136.)

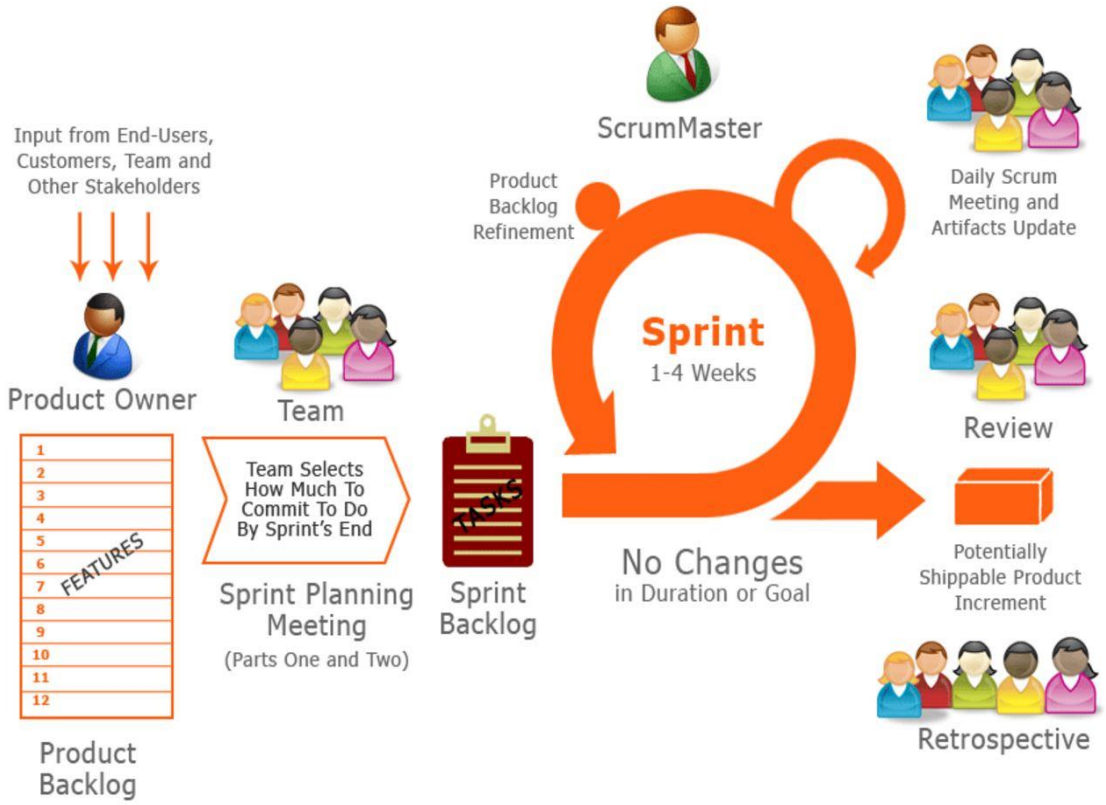
5.1 The Scrum Theory

Empirical/experimental process control theory is the base of Scrum. Empiricism refers that the experience generates knowledge and making decisions based on what is known. Scrum assigns an incremental, iterative approach to control risk and optimize predictability. Transparency, inspection, and adaptation are the three pillars which uphold each implementation of empirical process control. Important aspects of the process should be visible to those responsible for the outcome. Transparency needs those aspects be addressed through a common standard so observers share a common knowledge of understanding of what is being observed. Scrum artifacts and progress toward a goal must be inspected by Scrum users to find out undesirable variances. The inspection should not be so frequent; inspections are most fruitful when diligently executed by proficient inspectors at the point of work. (Schwaber et al. 2012, 137.)

When any deviation of one or more aspects of a process is determined by the inspector and if the resulting product is not acceptable then the process or the processed material must be adjusted. A quick adjustment must be done to reduce further deviation. There are four formal opportunities prescribed by Scrum for inspection and adaption: Daily Scrum**,** Sprint Planning Meeting**,** Sprint Retrospective**,** and Sprint Review**.** (Schwaber et al. 2012, 138.)

5.2 The Scrum Team

The Scrum team consists of the Development Team, a Scrum Master, and a Product Owner. The teams are known as self-organizing and cross-functional. The team members of a self-organizing team select their best working strategies and they are not directed by someone else from outside. (Schwaber et al. 2012, 138)



Graph 24: Function of Scrum Team (Agile Buddha 2015.)

Cross-functional teams offers all competencies needed to execute the task without relying on others not part of the team. Scrum's team model is designed to optimize creativity, productivity, and flexibility. Products are delivered iteratively and incrementally by the Scrum teams. (Schwaber et al. 2012, 138.)

5.2.1 The Product Owner

The Product Owner is responsible to maximize the value of the work of the Development Team and the product. It is only the Product Owner who is responsible for managing the Product Backlog. Management of Product Backlog includes: product backlog items are clearly expressed; ordering the item in the Product Backlog to best achieve goals and missions; ensuring the work performance value of the Development Team; making sure that the Product Backlog is transparent, visible, and clear to all, and determining the next task that Scrum Team will perform; and making sure that the Development Team is aware about the items in the Product Backlog to the level needed. (Schwaber et al. 2012, 138.)

Either the Product Owner or the Development Team may do the above mentioned works, but the Product Owner is still the accountable for these tasks. The entire organization must show respect on the decisions of the Product Owner, so the Product Owner can be succeeded. There is reflection of the Product Owner's desires/decisions in the content and Product Backlog ordering. It is not allowed that someone else would force the Development Team to perform from a different set of requirements. (Schwaber et al. 2012, 139.)

5.2.2 The Development Team

A Development Team is consisted with a group of professionals who perform the tasks of delivering a potentially releasable Increment of ready product after each Sprint. The Increment is created by only Development Team members. It is structured and empowered a Development Team by the organization to organization and handles own work. A

Development Team's overall effectiveness and efficiency are optimized by the resulting synergy. Development Teams are self-organizing. No one is allowed to tell them how to turn the Product Backlog's functionalities. The development team is cross-functional. There are no titles for Development Team members except developers, regardless of the task being executed by the person. Though the individual team members may be special for particular skills and areas, but it is the whole Development Team who is accountable for the deeds.There are no sub-teams under the Development Teams. (Schwaber et al. 2012, 139 – 140.)

5.2.3 The Scrum Master

It is Scrum Master who is responsible to ensure that Scrum is understood and enacted. However, to do these jobs Scrum Masters must ensure that the Scrum Team follows to Scrum theory, rules, and practices. The Scrum Master helps the Scrum Team to understand which interactions are helpful for the Scrum Team and which are not. The Scrum Master assists everyone to maximize the value which is created by the Scrum Team. (Schwaber et al. 2012, 140.)

The Product Owner is served by the Scrum Master in various ways, like: the Scrum Master helps the Product Owner by finding effective management techniques for Product Backlog. Scrum Master teaches the Scrum Team to make clear and concise Product Backlog items for the Product Owner. Clearly communicating goals, visions, and Product Backlog item to the Development Team. Long-term product planning is understood by the Scrum Master in an empirical environment. The Scrum Master helps in practicing and understanding agility**.** Scrum Master facilitates Scrum events as needed or requested. (Schwaber et al. 2012, 140 – 141.)

The Development Team is served by the Scrum Master in various ways, like: the Scrum Master helps the Development Team to be self-organizing and cross-functional. The Scrum Master leads and teaches the Development Team to develop high-value products. The Scrum Master helps to ensure the Development Team's progress by eliminating

impediments. The Scrum Master facilitates Scrum events as needed or requested**.** The Development Team is coached in organizational environments in where the Scrum is not yet completely understood and adopted. (Schwaber et al. 2012, 141.)

The organization is served by the Scrum Master in various ways, like: the Scrum Master leads and coaches the organization for Scrum adoption. He/she Makes plan for Scrum implementation. The Scrum Master assists stakeholders and employees to understand and enact Scrum. Causing change that enhances the effectiveness of the Scrum Team. The Scrum Master works with other Scrum Masters to enhance the efficiency of the Scrum application in the organization. (Schwaber et al. 2012, 141.)

5.3 Scrum Events

In Scrum the referred events are used to create regularity and to reduce the need of unnecessary meetings which are not mentioned in Scrum. There are time-boxed events in Scrum so that each event gets a maximum duration. This makes sure an appropriate amount of time for each event and there is no waste of time in the planning process. In Scrum, the Sprint is like a container for all other events and each event is a formal opportunity to supervise and adapt something. (Schwaber et al. 2012, 141 – 142.)

5.3.1 The Sprint

The Sprint is like the heart of Scrum, it is a time-box of one month or less where a ready, useable, and releasable product increment is developed. A brand new sprint starts forthwith after the ending of the previous Sprint. The key parts of Sprints are: Daily Scrums, Sprint Planning Meeting, the Sprint review, the development work, and the Sprint Retrospective. During the Sprint, the changes which may affect the Sprint goal are not made; the

composition of Development Team stays constant; and no decrementing of quality goals. (Schwaber et al. 2012, 142.)

Each Sprint is considered as a small project with maximum time duration of one month. Sprints are used to execute something just like projects. Each Sprint contains a definition about what is to be developed, a design and proper plan that would guide developing it, and the resultant product. Time duration of Sprint is limited with one calendar month. The definition of what is to be built may changes in case of long Sprint duration. Moreover, there is chance to increase the risk and complexity. (Schwaber et al. 2012, 142.)

It's possible to cancel a Sprint before the Sprint time-box is over, and it's only the Product Owner who is authorized to cancel the Sprint. When the Sprint goal becomes obsolete then the Sprint would be cancelled. This might happen if the company alters policy/direction or if technology or market conditions alter. The completed and ready Product Backlog items are reviewed after cancelling a Sprint. When the portion of the task is potentially releasable, typically the product is accepted by the Product Owner. The Product Backlog items are re-arranged and set back on the Product Backlog which is incomplete. It needs to regroup and manage another planning meeting for starting a new Sprint after cancellation a Sprint; as a result it consumes resources cancellation a Sprint. Actually Sprint cancellation is very uncommon and traumatic to the Scrum Team. (Schwaber et al. 2012, 142 – 143.)

5.3.2 The Sprint Planning Meeting

All the tasks what would be performed in the Sprint are planned first during the Sprint Planning Meeting. The sprint planning meeting is conducted by the Scrum Master, the Product Owner, and the entire Scrum Team. The stakeholders from outside may attend in the meeting, but it is very rare case in most companies. (Schwaber et al. 2012, 143.)

The Product Owner point out and describes the prioritized features to the Scrum Team during the planning meeting. The team and the Product Owner negotiate about the stories

that would be tackled by the team in that Sprint. The conversation between the team and the Product Owner is a time-boxed conversation. It is the Product Owner who is the responsible to determine the highest prioritized stories to the release, but the team is authorized to push back and reveal their own opinions. The corresponding user stories are added into the sprint backlog by the Product Owner once the team agrees to handle the work. (Schwaber et al. 2012, 143.)

5.3.3 Daily Scrum

The development team arranges a 15-minute time-boxed event in every day to synchronize tasks and make a working plan for the next 24 hours. Every day at the same time and place the Daily Scrum is held. It is a very effective way to reduce complexity. Each team member answers the following questions during the meeting: What has been done since the last meeting? What would be accomplished before the next Scrum meeting? And what impedes me from executing my task as well as possible? (Schwaber et al. 2012, 145.)

The Daily Scrum is used to analyze progress towards the sprint goal and to analyze how progress is trending toward executing the task in the Sprint Backlog. After the Daily Scrum, the Development Team meets to make a plan for rest of the Sprint's task. The Development Team should be able to explain the daily progress to the Scrum Master and Product Owner. (Schwaber et al. 2012, 145.)

The Daily Scrum is conducted by the Development Team but it is the Scrum Master who ensures that there is the meeting for the Development Team. The Development Team gets lessons from the Scrum Master about how to maintain the Daily Scrum within the 15-minute time-box. The Daily Scrum is a significant event in Scrum development; it improves communications, reduces unnecessary meetings, inspects and removes obstacles to development, ensures quick decision making, and improves the knowledge level of the Development Team. (Schwaber et al. 2012, 146)

5.3.4 Sprint Review

The objective of Sprint Review is to inspect the increment and adjust the Product Backlog if needed. A Sprint Review is occurred at the end of the Sprint. During the Sprint Review, the stakeholders and the Scrum Team collaborate about what was accomplished in the Sprint. (Schwaber et al. 2012, 146.)

The included elements of Sprint Review are: what has been accomplished and what has not been accomplished are identified by the Product Owner. The Development Team identifies what went fine during the Sprint, what impediments it ran into, and the ways those impediments were eliminated. The executed work is demonstrated by the Development Team and they explain the answers of the questions about the Increment. The Product Owner negotiates about the existing Product Backlog. The entire group contributes to determine the next step, so that the subsequent Sprint Planning Meetings get valuable input form the Sprint Review. The Sprint Review's result is known as the revised Product Backlog which demonstrates the Product Backlog items for the subsequent Sprint. (Schwaber et al. 2012, 146.)

5.3.5 Sprint Retrospective

For the Scrum Team, the Sprint Retrospective is a chance to inspect itself and make a plan for improvements to be enacted during the subsequent Sprint. And it occurs after the Sprint Review and before to the subsequent Sprint Planning Meeting. The Sprint Retrospective is a time-boxed meeting with three hours for one-month Sprints. However, proportionally less time is assigned for shorter Sprints. (Schwaber et al. 2012, 147.)

The objectives of the Sprint Retrospective are: detect how the last Sprint went with regards to relationships, people, process, and tools; identifying and ordering the major items which went well and potentially improved; and developing an effective plan for the Scrum Team so that the Scrum Team can operates its works flexibly. (Schwaber et al. 2012, 147.)

The Scrum Master motivates the Scrum team so that the Scrum Team can improve within the Scrum process framework, and create its work enjoyable and effective for the subsequent Sprint. The Scrum Team creates plan about how to develop the product quality during each Sprint Retrospective. Actually the improvements that would be implemented in the next Sprint should be identified by the Scrum Team by the end of the Sprint Retrospective. (Schwaber et al. 2012, 147.)

5.4 Scrum Artifacts

Scrum artifacts define work or value in different ways which are effective in providing opportunities and transparency for inspection and adaptation. Scrum's defined Artifacts are designed specially to increase transparency of important information needed to make sure that the Scrum Teams are successful to deliver an accomplished Increment. (Schwaber et al. 2012, 147.)

5.4.1 Product Backlog

Product Backlog is a list of prioritized features and this list contains information of all functionality needed in the product. The Product Owner is the sole person who is responsible for the Product Backlog. It is dynamic and never complete. The Product Backlog constantly changes to detect what the product needs to be competitive, appropriate, and useful. The Product Backlog exists, as long as a listed product exists. (Schwaber et al. 2012, 148.)

The Product Backlog contains all functions, features, enhancement, requirements, and creates the plan that would be applied to the product for any changes in future releases. The items of Product Backlog contain the attributes of an order, description, and estimate.

The Product Backlog is ordered by priority, risk, value, and necessity. The Product Backlog items which are top-ordered are taken into account to develop first. (Schwaber et al. 2012, 148.)

Among the Product Backlog items, the top-ordered items are more detailed and clearer than lower-ordered ones. The Product Backlog items that would be taken into action in the upcoming Sprint by the Development Team are fine-grained, having been decomposed so that any one item can be accomplished within the Sprint time-boxed. Changes in the technology, market conditions, or business requirements may cause changes in the Product Backlog. (Schwaber et al. 2012, 148 – 149.)

The Product Backlog grooming is an on-going process in which the Development Team and Product Owner collaborate on the details of Product Backlog items. It is the act of adding information, estimates, and order to items in the Product Backlog. The items are reviewed and revised during the Product Backlog grooming. But it is possible to update the items at any time and only the Product Owner can perform this. The Scrum team decides how and when the grooming would be executed. During the Sprint, the grooming is a part-time activity between the Development Team and the Product Owner. (Schwaber et al. 2012, 149.)

5.4.2 Sprint Backlog

The Sprint Backlog is known as the set of Product Backlog items that are chosen for the Sprint along with a plan of delivery of the product increment and realizing the Sprint goal. It is a forecast about the type of functionality that would be taken into account in the subsequent Increment and the tasks needed to deliver that functionality; and this forecast is made by the Development Team. The Sprint Backlog refers the set of works what would be performed by the Development Team to turn Product Backlog Items into an accomplished Increment. Moreover, the Sprint Backlog makes the work visible which is identified by the Development Team and this work is needed to achieve the Sprint goal. (Schwaber et al. 2012, 149 – 150.)

Throughout the Sprint the Sprint Backlog is been modified by the Development Team, and it emerges during the Sprint. The Development Team adds the work to the Sprint Backlog whenever it needs to add new work. As work is accomplished or performed, the estimated rest of the work is updated. The plan's elements are removed whenever it deems unnecessary. During a Sprint, only the Development Team can alter its Sprint Backlog. It is highly visible a Sprint Backlog, and it is the real-time picture of the task that the Development Team intends to execute during the Sprint, and it belongs only to the Development Team. (Schwaber et al. 2012, 149 – 150.)

5.4.3 Increment

The concept of increment is very important in agile software development. In Scrum, the maintenance of increment is the key of a project success. The sum of the completed Product Backlog items during a Sprint and all previous Sprints is known as the Increment. Whenever a Sprint ends a new Increment must be done. (Schwaber et al. 2012, 150.)

6 CONCLUSIONS

This thesis explains all important aspects of agile, and provides theory about what agile is and what is contains. There is bunch of information of all important agile methods and practices which helps to evaluate and select the best agile methods and practices for a development team. The survey part demonstrates all practical information of agile practices in current world. Moreover, the survey shows that the traditional methods are no longer effective and agile methods are getting popular day by day. Scrum is the most popular method among the developers and organizations. This thesis provides a complete guideline about Scrum which is effective for selecting the Scrum as the software development method.

There are different types of agile methods and practices for software development. They are not equal in strength, flexibility, weakness, risk, and usage. Scrum is a lightweight development method with long sprints and its popularity is growing day by day. Extreme Programming provides a robust methodology for release planning, and it is considered to be low risk. Many developers like to combine practices from Scrum and Extreme Programming and apply them in software development.

Feature Driven Development method has some flexibility and this method is based on a traditional object-oriented design method with good quality controls. Crystal Methods offers a good blend of agile practices with only moderate risk. It has a large number of traditional and agile practices. It had been designed to be scalable. Dynamic System Development is still very popular in United Kingdom. This method is product focused with wide range of practices. The practices are with low risk but not flexible. Adaptive Software Development is especially for complex and large software development project. In this development method, the project is carried out in cycle.

Software engineering is a disciplined, systematic, and quantifiable approach for developing software. Similarly, agile methods are disciplined, systematic, and quantifiable approach

for building software products. Agile methods adapt the values of customer collaboration, adaptability, and iterative development to change to the development of software products to satisfy customers.

A very important thing we can learn from this thesis that agile methods should not be judged on the basis of their size or how much they resemble to traditional methods. Rather agile methods should be judged on the basis of strength of their individual practices. For example, the release planning methodology of Extreme Programming is the best among all agile methods and this method contains some key practices as well. Large sized agile methods may enhance risk as it is challenging to adopt a large number of practices in software development process. Moreover, large never guarantees low risk. Therefore, just-in-time, just-enough, and right-sized agile practices with adaptable and flexible process may be effective for a development team.

REFERENCES

Ambler, Scott 2002.
Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process. New York: John Wiley & Sons, Inc.

Ambler, Scott W. 2005. Feature Driven Development (FDD) and Agile Modelling, Available: http://www.agilemodeling.com/essays/fdd.htm. Accessed: 5th August 2015.

Agile Buddha, Available:
http://www.agilebuddha.com/trainings-workshops/scrum-training-workshop/.Accessed: 7th October 2015.

Clifton and Dunlap 2003. What Is DSDM, Available:
http://www.codeproject.com/Articles/5097/What-Is-DSDM.
Accessed: 14th July 2015.

Cockburn, Alistair 2002.
Agile Software Development. Boston: Addison-Wesley.

Coram and Bohner, 2005. The impact of agile methods on software project management, Available:
http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1409937&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F9677%2F30561%2F01409937.pdf%3Farnumber%3D1409937. Accessed: 22nd July 2015.

Highsmith, Jim 2000. Retiring lifecycle dinosaurs –Using Adaptive Software Development to meet the challenges of a high-speed, high-change environment, Available:
http://www.adaptivesd.com/articles/Dinosaurs.pdf. Accessed: 16th September 2015.

Jeffries, Ron 2011. What is Extreme Programming, Available:
http://ronjeffries.com/xprog/what-is-extreme-programming/. Accessed: 26th October 2015.

Lawton, Richard 2015. Feature Driven Development: A Guide, Available:
http://www.arrkgroup.com/thought-leadership/feature-driven-development-a-guide/.
Accessed: 11th August 2015.

Larman, Graig 2004. Agile & Iterative Development. A Manager's Guide. Boston: Pearson Education, Inc. Available:
http://upload.wikimedia.org/wikipedia/commons/thumb/5/50/Agile_Project_Management_by_Planbox.png/300px-Agile_Project_Management_by_Planbox.png. Accessed: 22nd June 2015.

Larman, Graig 2004.
Agile & Iterative Development. A Manager's Guide. Boston: Pearson Education, Inc.

Orr, Ken 2001. Adaptive Software Development, Available:
http://www.exa.unicen.edu.ar/catedras/agilem/cap23asd.pdf. Accessed: 15th October 2015.

Rico, David F., Sayani, Hasan, Sone, Saya 2009.
Business Value of Agile Software Methods: Maximizing ROI with Just-In-Time Processes and Documentation. J. Ross Publishing Inc.

Schuh, Peter 2005.
Integrating Agile Development in the Real World. Massachusetts: Charles River Media, Inc.

Schwaber, Kubacki, and Sutherland 2012.
Software in 30 Days : How Agile Managers Beat the Odds, Delight Their Customers, and Leave Competitors in the Dust. John Wiley & Sons

Versionone 2015. Available:
https://www.versionone.com/pdf/state-of-agile-development-survey-ninth.pdf. Accessed: 12th November 2015.

Waters, K., 2007. What Is Agile? (10 Key Principles of Agile), Available:
http://www.allaboutagile.com/what-is-agile-10-key-principles/. Accessed: 18th June 2015.

Wells D., 1999. Iterative development, Available:
http://www.extremeprogramming.org/rules/iterative.html. Accessed: 17th June 2015

Wells, D., 1999. The Rules of Extreme Programming, Available:
http://www.extremeprogramming.org/rules.html. Accessed: 16th September 2015.

Wisegeek, 2015. What is iterative development, Available:
http://www.wisegeek.com/what-is-iterative-development.htm. Accessed: 4th July 2015.