Purushottam Thapa Magar

Software Development Process in Small Enterprises

An insight into distributed development

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Degree Programme in Media Engineering

Bachelor's Thesis

4 September 2013

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Purushottam Thapa Magar Software development process in small enterprises An insight into distributed development 78 4 September 2013
Degree	Bachelor of Engineering
Degree Programme	Media Engineering
Specialisation option	JAVA and .NET Application Development
Instructor(s)	TeemuPiirainen, CEO of DooxeOy Kari Aaltonen, Principal Lecturer

The aim of this study is to analyse the overall work flow and processes that takes place in small enterprises during software development and give an insight into the distributed approach in software development.

There are thousands of small sized software companies all around the world. Naturally, small firms possess fewer resources and manpower than bigger companies like Microsoft and Oracle. However, the contributions made by these firms to the software industry are outstanding.

The world has turned into a global village. Information Technology is the father of this remarkable achievement of mankind. Hence, it is obvious that this field benefits the most from it. One of the advantages is that the software development can be distributed in various geographical locations, which is a huge leap from the traditional development methods. The possibility of distribution of development team has given a birth to a new kind of era, which is known as software outsourcing. Outsourcing is a very effective method to employ for rapid and creative development. In fact, it provides huge financial savings to businesses. Nevertheless, outsourcing also creates problems, which are irrelevant in centralized development. Those probable problems should be considered beforehand or it will jeopardize the entire project.

Keywords	DSD, dooxe, scrum, agile development, version control, FDD, Extreme Programming.

Contents

Abbreviations and terms

DSD	Distributed Software Development
SDLC	Software Development Life Cycle
TDD	Test Driven Development
XP	Extreme Programming
FDD	Feature Driven Development
CMS	Content Management System
MVC	Model-View-Controller
VCS	Version Control System

1 Introduction

The purpose of this thesis is to study the distributed aspect of software development methodologies in small companies. Moreover, it intends to study the role of developers in small firm's working environment.

The United States of America alone has almost one million software developers [1]. We can do the math to guess the number worldwide. The demand and need of new software is never going to stop in today's world. As a result, the number of software companies has increased exponentially in the last one decade. However, the majority of these enterprises are small sized businesses. It is very interesting to study a development process in small firms because they are very versatile and frequently follow a modified version of standard development methods. This is an effective method, which allows changes to be executed with less or no amount of extra work. They work in small teams and usually inside the same room, which makes the adaptation fairly simple.

On the other hand, software development has entered into a new era. Nowadays, development does not occur in one geographical location. Many development teams work in a same project from various locations of the world. Such approach to software development is known as Distributed software Development (DSD). It is one of the important aspects of today's software industry. Normally, DSD comes into play when companies start to look for better opportunities, either financial or qualitative.

The study is based on the web services called dooxe and korjausurakka. Both applications are reverse auction platform and share the same backend. The only difference between the two is the appearance and the group of users. The idea behind the application is very simple, service seekers simply post a job in the service and contractors try to outbid each other by offering their best offer. Finally, service seekers select the service provider that best suits their need. Customers can post any kind of job in dooxe whereas korjausurakka is strictly developed for a job relating to construction.

Dooxe oy owns both services mentioned above and commissioned this thesis. It is a very small firm based in Helsinki. The dooxe team consists only six members. It is not a

software company but a company that operates a web service. However, it does not order other software companies to develop and maintain their application.

A meticulous presentation of how software is developed in dispersed environment in small firms is the aim of this paper. Dooxe is a very suitable choice for such study because first of all the only developer in dooxe team lives in Switzerland and second, it is a very small company.

2 Basics of Software Engineering

The software industry has been through drastic changes over last three decades. At the early stage of software industry, computer systems were not very complicated but really simple and light. As a matter of fact, their capacity was so limited that they could hardly handle more than one task at a time. 30 years ago, machines fitted out with single processor used to run computer applications. Software on those days used to take input from a single device. Keyboards were the most common source for inputting data into the application. Outputs of the applications were either letters or numbers. Today, computer applications are very complex and are often based on server-client model. Applications have graphical user interfaces and are capable of receiving inputs from multiple devices. Computer applications are so advanced that they can take inputs from sensors or even satellites. Mobile applications are a very good example of sensor driven software. Furthermore, they do not just produce alphanumeric outputs anymore. Those are complex results that are not understood by ordinary people. Outputs can be in different forms such as images, 3D models, old school alphanumeric values or everything combined. Nowadays, software runs on a machine with multiple processors. In addition, such machines can be geographically scattered or running a different operating systems. [2,2.]

Software has become crucial in all areas of human life. At present context, programming skill alone is not good enough for producing quality software. Software industry has seen serious problems such as over cost and late delivery of software. Moreover, the quality of software products has been compromised. Similarly, there is no effective mechanism for the maintenance of software. These are worrying issues that need to be addressed. [2,5.]

The concept of software engineering was initiated to solve all the problems that were mentioned above. The key objective of software engineering is to develop maintainable and excellent software within given time and budget. A disciplined protocol is followed to acquire these objectives, which are called software development methods. [2,5.]

2.1 Some terminology

For the ease of readers of this paper, terms that are frequently used during software development are discussed in this section. In software engineering, there are many

expressions that need to be explained to a novice person. However, many terms are beyond the scope of this paper. Thus, only terms that are relevant to this thesis are discussed below.

Deliverables

Deliverables are the things that are produced during development. Examples of deliverables would be source code, database model, entity relationship diagrams, user manuals, sequence diagrams and operating manuals. Each of these items is called deliverable in software engineering.

Milestones

In software engineering, milestones are points that are used to define the completion of different phases of a development work. As an example, completion of database design would be like reaching one milestone or simply, the whole process is a milestone. Accomplishment of user interface design would be another milestone. Hence, milestones are nothing but events, which are utilized to monitor the progress of the project. [2,12.]

Product

The collection of deliverables that is delivered to a client is known as product. In simple words, it is a package that contains all the deliverables that were generated during development. [2,12.]

Process

Process is a method that is followed to develop software. It is used to ensure a quality of software. Different companies follow different process. Some of the examples are scrum, velocity tracking, waterfall, feature driven development and crystal clear. [2,12.]

2.2 Misconceptions about software

There are many fallacies among people, which directly concern software industry. A few of them are very critical because they directly affect the software development process. [2,4.] They are discussed below:

Modification of software is easy

In programming, one task can be implemented in several different ways; hence change of code is easy. However, it is absolutely wrong to mix this idea with modification of entire software. Software follows certain design and architecture, which needs to be planned cautiously at the preliminary phase. Software architecture is like a foundation of concrete building: if it is created once then it is extremely difficult to change. Once software architecture is written, an entire application needs to be written in a way that it goes along with the pre-written architecture. [2,4.]

On the other hand, change is not completely impossible. Software can be modified but it requires tremendous amount of time and money. Change is less expensive if development process is in initial phase. Moreover, it is less time consuming. [2,4.]

Software testing eliminates all the errors

This is another myth that people have about software. Software testing does not necessarily remove all the errors. The objective of software testing is to find the presence maximum possible errors. However, it does not guarantee the absence of errors. The test results depend on the way that test cases are written. More testing will definitely provide a safer system but it still cannot promise the absolute accuracy of software. [2,4.]

Software is 100% accurate

It is a very traditional and widely accepted view that software makes no mistakes. In a way, it is true but there are limits and exceptions to everything. Computer software does produce false results occasionally. Because software testing cannot guarantee bug free application, there is always a little window for mistakes to happen. However, well-designed software does it very rarely. [2,4.]

Software engineering does not require prototypes

Almost every engineer first develops a prototype before putting an object into production. Nevertheless, people think that computer software does not require that kind of prototyping. They have this preconceived idea that software can do the job right in the very first time. This is absolutely wrong: if that would have been right, the concept of beta versions and testing would never have been born. Despite this fact, it is very hard to convince customers, thus software engineers accept the job that is against the software developing principles. [2,4.]

More the feature, better is the software

It is a normal human behavior that people want more. Perhaps, that is the reason why they want more and more features in computer applications. Normally, people think that software with massive list of features is better than the simple one. Actually, the exact opposite is the truth. In real life, software or any other device that is designed for one specific task does the job well. [2,5.] As an example, Facebook have lot more features than twitter. However, twitter much more efficiently does the task of delivering messages to wider range of people than Facebook. Therefore, many celebrities use twitter instead of Facebook.

Software does not require maintenance

Because software is not a physical object, a considerable amount of people may think that it never wears and tears. In traditional sense, it can be considered true. Nonetheless, modern concepts totally deny it. All computer software requires timely integration and maintenance. Technology is changing very fast, and what is working today might not work tomorrow. Hence, upgrading needs to be carried out in regular basis. Moreover, source code can change if software is handled improperly. Such change can result into crash or strange behavior of software. Proper maintenance is important to resolve those issues. Furthermore, performance of software might decline as the time on production increases. Abundance of old and useless data in database might be one reason for this kind of issue. Such problems will not arise if proper maintenance of software is done.

Obviously, the mistaken notions described above cause trouble in development work. In big companies like Oracle, it would be impossible to convince either management or customers about these issues. On the other hand, managing director of the company is often the member of development team in small firms. Hence, face-to-face meeting with the decision maker is possible and the situation can be explained thoroughly. In giant companies, management just wants to see the job done within deadline. However in small enterprises, decision of the boss can be influenced with proper reasoning.

2.3 General software development approach

Although there are many software development principles, the core of the software engineering is always same. All the existing methods follow this basic development pattern. All the phases of basic development framework will be discussed below.

Software Requirements

In software engineering, requirement gathering is the phase when features and functionalities of the software are discussed and written properly. It is the first phase in development cycle. Requirement gathering is a very complicated task because user needs are often hidden deeply within lots of assumptions and misconceptions. The purpose of software is to solve a problem or achieve an objective of a user. To solve a problem, user needs should be understood. A software engineer must not make assumptions of client's needs. A false assumption leads to a wrong solution, hence customers must be questioned again and again to identify the problem precisely.[3,105-106.]

Requirement gathering is a systematic process. It is performed in iterative cycles. At first, the problem is observed, and then it is documented. Later, acquired information are checked to assure the preciseness. The same process is repeated until all the requirements are documented properly. [3,106.]

In software development, most of the time the client does not know what his/her software needs are. They often present vague needs. It is the job of software engineers to show them the right path. In the requirement gathering phase, involvement of the client is inevitable. In fact, the development team should persuade customers to spend as

much time as possible in this phase. Inadequate attention from client can result into redoing of work. For instance, if the development team writes a software specification on their own, clients might disagree with that idea, which will lead to the complete waste of time spent and frustration of the team. Hence, customers must not skip the requirement analysis meetings.

Design

Designing is the second phase of software development life cycle. It is the phase when software specification is written. In design phase, software structure and architecture is defined [4,231]. Likewise, user interfaces and test plans are created. All the logical models for software are defined. Security measures are also designed in this phase. Similarly, inputs and outputs of the application are defined. In addition, if agile methods are used, testing of user interface is also executed in this phase. Primary testers are end users. End users provide lots of valuable data regarding user interfaces. However, professional testers are also used in interface testing.

Later in the designing phase, previously written user-oriented designs are converted into machine-oriented designs. Such computer-oriented designs are often called as system designs of software. Data structures and software modules are part of system designs. Further, coding conventions and database structure are defined in system design phase. [4,231.]

Coding

Coding is the phase when actual development of software takes place. It falls between designing and testing phase. Basically, it is just a technical implementation of designs created in previous phase. Coding conventions and data structures defined in software designs are strictly followed. [5, 18.] Moreover, developers themselves carry out unit testing in this phase. Deliverables generated in this phase are source codes.

Testing

Testing is a fourth phase of Software Development Life Cycle (SDLC). Testing basically refers to the functional testing of source code. User interface testing is not considered at this point because such testing has already been carried out in the design

phase. The main objective of functional testing is to find possible bugs. At this stage, professional testers are the primary testers. They test the application by following test plans precisely. [5,19-20.]

Also the end users perform functional testing in many occasions. Usually companies release beta versions of software to allow users to perform these testing.

Deployment

Deployment phase means the installation of source code for operational use. Source codes are placed in different environment depending on the nature of an application. Web applications are usually deployed in web servers whereas standalone applications are installed in machines. After deployment, software is ready for use.

Maintenance

Maintenance is the last and a never ending phase. Maintenance of software is required as long as the software is in production. In this phase, timely update and integration of plugins and supporting software is done. [3, 221.] Moreover, old and irrelevant data are archived to remove the unnecessary burden from an application. Applications may need additional resources as the time in production increases. Addition of such resources is also part of the maintenance cycle.

2.4 Why software engineering is required

Software crisis is the main reason why software engineering is required. Software crisis has been with us since the early days of software industry. Software industry is growing with lightening speed but complete eradication of problem has not been successful yet. Problems that we faced 40 years ago still exist. Software projects still surpass anticipated budget and time. In addition, they are still faulty. According to IBM report, 53% of the projects exceed the predicted budget by an average of 189%, Among every 100 projects, there are 94 restarts and worst of all 31% of the projects are never completed because they get cancelled in the middle. [2,2.] Next, the importance of software engineering will be reflected with historical examples.

Y2K crisis

In the year 2000, an unexpected software problem crippled computer systems globally. The problem was with a date format. At that time, dates were shortened to two-digit format. For example 1987 was written as 87. Software developers could not think of an exceptional condition of the year 2000. Millions of dollars were invested to fix this small problem. [2,2-3.]

Failure of Patriot Missile

Patriot is a defensive missile produced by “star wars” program of the United States. It was first used in the Gulf war. Patriot’s main objective was to shoot down incoming Iraqi Scud missiles. However, it failed several times. In one occasion, 28 American soldiers were killed in Saudi Arabia. The reason was a software bug. Small timing error was present in the system’s clock. After 14 hours of operation, tracking system was no longer accurate. At the time of the attack in Saudi Arabia, the system had been operating for more than 100 hours. [2,3.]

Crash of Ariane-5

In 1996, the first test flight of Ariane-5 space rocket crashed within 39 seconds of its launch. It was built in 10 years of time and a fortune of 7000 million dollars was spent. The problem was very small and simple. Guidance system’s computer tried to convert the value from one format to another and an overflow error occurred because the converted value was too big. Developers knew about the conversion but they simply ignored it because they assumed that the value would never be big enough to result in any critical failures. [2,3.]

Hence, software crisis does not just crash computers but can bring down a giant machine like a space rocket. There are many other failures that have caused loss of life and fortune. In order to prevent these catastrophic disasters, systematic rules and discipline are needed to develop quality software. Software engineering is a scientific discipline, which ensures good and quality performance of an application. Software engineering principles cannot guarantee 100% accuracy of software. However, it brings down the risk factors significantly and provides an acceptable level of assurance.

On the other hand, software engineering helps to keep the project within the estimated budget and deadline. In conclusion, software engineering principles are important and must be followed in all software development projects.

3 Software Development Methodologies

In simplistic terms, software development methodology is defined as the set of rules that are followed by developers during development of an application. It provides a framework for planning and controlling the entire software development process. [6,53.] A relevant set of specific software development practices creates a software development method. The term “specific practices” refers to all minute by minute tasks carried out during development. For example, writing use cases can be considered as one of the minute tasks in software development. A software development method must satisfy all the fundamental activities mentioned in section 2.3. A software development process basically converts user needs into a real life application. There are intermediate conversions involved. At first, user requirements are changed to software requirements. Then, software requirements are transformed into software designs. Coders implement the designs produced. After that, a testing team tests the software with all test cases that are likely to occur. Finally, software is deployed for operational use.

The main objective of software development methodologies is to prioritize the order of stages followed during development. Furthermore, it provides a standard for transiting from one stage to another. In simple words, it specifies things that need to be completed before moving forward to another stage. All development methodologies share the same fundamental software development model: requirement, design, code, test, deploy and maintain [6,53]. However, there are significant differences in the details of these stages. It is very important to understand that all methodologies satisfy the basic software development model but are different from each other in minute level.

As mentioned earlier, software products are no more mere calculating programs, but rather complex systems. Such complexities are skyrocketing, as the users are being more and more demanding. [2,2.] Today, there are thousands of software products floating around the market. However, very few of them are advantageous. Useful software is a result of systematic development process. Moreover, they evolve over time to meet the needs of users and changing environments.

Computer applications were very expensive at the early stage of software industry. Furthermore, they required good computer proficiency to produce fruitful output. Hence, the number of users was very small. Time has changed and millions of people are using all sorts of computer software. Today, the types of computer software have

no limits. They can be simple numeric calculators or advanced 3D games. Amazingly software with different purposes are being developed everyday. Development of newer type of software is not slowing down but increasing with lightening speed. Nowadays, there is even software for measuring alcohol level in your blood. Basically, it calculates how drunk you are. Due to all these facts, users of computer applications have increased from a few thousands to millions in the past years. In order to address this demand, more and more developers are being involved into a software development. Such a massive engagement from both users and developers side has made the development process very intricate. Thus, writing applications in a closed room with self-defined standards and procedure is not practical anymore. In fact, a well-defined and systematic rule is needed to manage such an intense project to get the decent outcome. Hence, software development methodologies come into play.

Software development process has evolved outstandingly to cope with the changing needs. Thus, there is a fairly good number of methods that are battle tested and proven to be efficient. At the present context, there are rarely any companies that do not follow these standard processes for their software development projects. Therefore, it is surely a good idea to get acquainted with the idea behind software development methods.

In conclusion, methodology is very important for software development. Many scholars have done a variety of researches and several software development methods have been developed. However, each of them has their pros and cons. Some of the popular ones are explained in the following sections of this study.

3.1 Traditional software development methodologies

The core idea behind traditional software development methodologies is that all the necessary information about the project is well known beforehand. Furthermore, that information is assumed to be stable and unlikely to change during course of development. [7,3.] Due to prior understanding of project's details, a good plan is laid out to complete the project. Hence, in software development community, the term plan-driven or heavyweight methodology is also frequently used for traditional method [8,47]. In software engineering, the longer the problem exists, the harder it is to get rid of it. Moreover, the solution might be quite expensive and tedious. For example, if there is any bug relating to a database, it is lot more difficult to remove it after one year than

right away because there will be hundreds of new entries piled during one year. More data means more time for their migration. Furthermore, it is risky because data can be lost or damaged during transfer. Therefore, traditional software development approach argues that detailed planning of architecture and design of software can minimize the cost and trouble of the project. The theory insists on prevention rather than fixation [8,47].

Like any other software methodologies, plan-driven methodologies also follow the same fundamental software development practices. First of all, requirement of the software is defined. In this phase, the length of different phases and the whole project is estimated. Furthermore, probable issues that may arise during the course of development are discussed and preventive measures are planned. The second step is to create designs and lay down an architectural plan. Diagrams are used to represent designs and architectural planning. They provide technical infrastructure to the project and create a path for implementation.

The third step is to implement an application for real. Coders write source code based on predesigned architectures. In the coding phase, work is divided according to the skill level of developer. In most cases, coding and testing are carried out simultaneously to figure out the issues as soon as possible. Stakeholders usually participate in final testing sessions. Software is delivered when the client is satisfied with the output. Delivery might just be the deployment in proper environment or supply of all the deliverables that were produced during the project. Due to these heavy phases, traditional practices are also known as heavyweight methodologies.

In heavyweight methodologies, the quality of predetermined information affects the success of project. It promotes the idea “Do it right the first time” because it denies any changes in the middle of the project. These kinds of software development models are outdated. However, some projects can still apply and enjoy their benefits. Typically, traditional methodologies are appropriate for software that handles sensitive information because any changes or problems are unwelcome due to safety reasons. Additionally, simple and small projects can make use of heavyweight methods. Some of the traditional methods are described below.

Code and Fix

Code and Fix is the oldest and most simple software development method. Although, it is the most impractical method available, developers still use it. The idea behind code and fix is very straightforward. Firstly, the developer comes up with a rough idea about a system. Then, implementation of that idea is carried out right away. As the development progresses, problems reveal themselves and solutions are developed accordingly. Despite the fact that code and fix welcome redundancy and failure, it still exists in wider extent. Nonetheless, it should be kept in mind that organized and serious software development has abandoned code and fix approach longtime ago. Currently, individual projects are the main scope area for code and fix. [9, 16.]

One of the reasons why code and fix method is still prevailing is that it does not require any expertise to make use of it. Furthermore, it is a method that totally goes along human nature which is preference over doing than thinking. Thus, code and fix is also known as “cowboy coding”. However, avoiding proper planning can be costly. In code and fix approach, it is not surprising if things need to be done all over again. In addition, extensive redesign of an application is a standard process developers go through. Another reason for the popularity of the code and fix method is instantaneous results. As an example, if there is a need of division function, writing a function directly is a lot easier than thinking through all the cases for that operation. As a result, peculiar cases like division by zero is not handled properly but function can be fixed or integrated as the problem arises.

Code and fix resembles hit and trial method. Developers use it all the time. In huge projects, code and fix is embedded in other SDLC models such as scrum. A project can be managed using another efficient model but developers use code and fix to complete parts of their task, sometimes even without realizing it. The division example explained above is a good illustration of embedding the code and fix method. Division function itself is not a project but it can be a part of project, for instance calculator application undoubtedly requires division function. Hence, calculator application is built using efficient SDLC model (for example, scrum). However, an individual developer can use code and fix for writing a little portion of it (division function).

Waterfall

Waterfall is one of the most popular classical software development models. Win Royce introduced it in the 1970s [8, 48]. The first waterfall model is known as the classical waterfall model, which had no kind of iteration whatsoever. However, the model itself has evolved and changed over time. As a result, new waterfall models provide some sort of iteration. [10,95] At the present context, hardly anyone follows the classical waterfall model.

The waterfall model was a great leap for software development industry when it was first introduced. It is a linear method that discards any feedback between different stages of software development life cycle. In waterfall development, series of sequential steps are carried out from very beginning to the end of the software development lifecycle. The order of such steps is very critical and also the most distinctive feature of waterfall model. In waterfall approach, next phase cannot start unless the previous one is completed. There is no overlapping between the different phases. Moreover, testing is discarded and only performed after the implementation phase. Thus, testing of the whole product rather than an individual component is encouraged. Massive test cases might disorient the testers and some vital test cases may go unnoticed. Such product is bound to face problems in future. Such approach also leads to the hard and time consuming debugging. [10,95-96.] In modern software testing, even small and obvious tests are recommended to be performed thoroughly.

In each step of development, the outcome is compared against requirements that were decided at the very first phase. Each phase is marked as done when all previously

set requirements are met. In addition, the product's quality is controlled in each and every phase. The image below depicts the classical process of waterfall development.

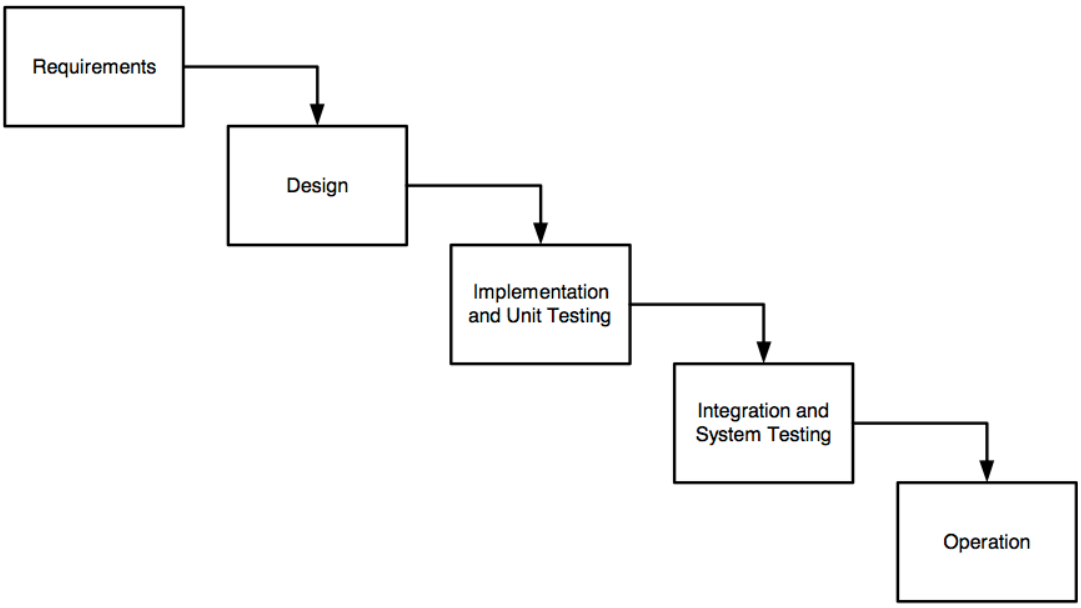

Figure 1.Classical waterfall development. [6,56]

Figure 1vividly illustrates the step-by-step development process. As depicted in figure 1, waterfall model also follows the general software development pattern. At first, all user requirements are listed and corresponding system architecture and layout is designed. Later, coders implement such designs. Then, testing is carried out to guarantee the efficiency and usability of an application. Finally, application is deployed and kept under continuous maintenance.

Waterfall is the oldest well-known software development model. The biggest flaw in waterfall model is that there is no way a project can be changed once the requirements are finalized. However, requirements are always changing during the long software development life cycle. Therefore, waterfall is a bad choice for big projects where requirements cannot be fully analyzed beforehand. On the other hand, systems that are well defined and straightforward are very good candidates for waterfall model. [11,18.] For example, development of a current location finder application is a simple and straightforward project. The only thing needed is to "find the right current location".

Apart from rejection to change, there were evidently other weaknesses of waterfall models. One of those shortcomings was the disobedience of strict sequential development advocated by waterfall model, which is basically the soul of the entire model. [8,49-50.] Such lack of obedience can be easily observed in the table below. The 'cod-

ing' column represents the amount of effort put on implementation and unit testing together.

Activity	Phase			
	Design	Coding	Integration testing	Acceptance testing
Integration testing	4.7	43.4	26.1	25.8
Coding	6.9	70.3	15.9	6.9
Design	49.2	34.1	10.3	6.4

Table 1.Division of work effort in different phases. [8, 49]

As shown in table 1, only 43% of the total coding has been carried out in coding phase. About 5% was done before the finalization of software design. Even worse, more than 50% of actual implementation was accomplished in testing phase, at which point application was suppose to be ready by all means. The result depicted in table 1 is against the very ethics proposed by waterfall model. Hence, strict enforcement of waterfall model is unrealistic. Software development process can be better defined as an opportunistic process, where developers move back and forth in different life cycle phases. In software development, it is quite normal to cross milestone boundaries of different phases.

V-shaped

As similar to waterfall model, V model use the sequential development approach. Moreover, development phases must not overlap each other, that is, the first step must be completed before the commencement of next one. Nevertheless, unlike waterfall model, V model focuses on testing. Test plans are formulated for each

phase. Once the implementation of a system is accomplished, all of the pre-written tests are conducted. [10,97-98.] Figure 2is the pictorial demonstration of V-shaped model.

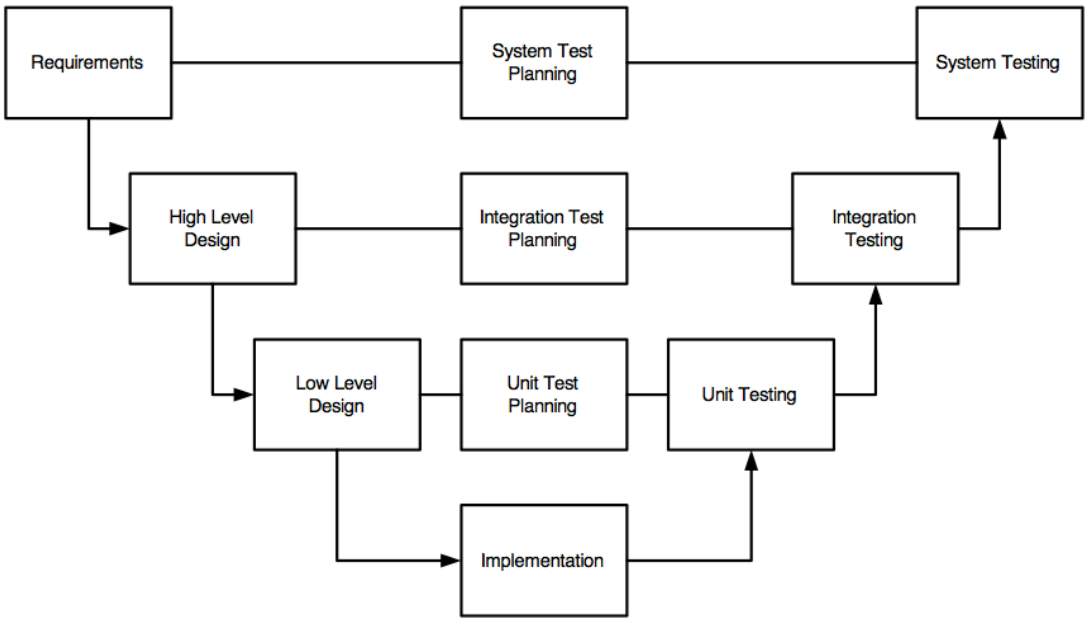

Figure 2. The V-shaped model.

As illustrated in figure 2, in V-shaped model, development phases that come before implementation phase are listed in the left-hand side and testing phase is broken down into several other phases and listed in right-hand side as it can be observed in the figure above. Such placement creates the shape of letter ‘V’, from which the model gets its name. After that, test plans that are written after the completion of each of the phases that lie in the left-hand side are placed in the middle of the V structure. As in any other SDLC model, gathering of requirements is the first step. Then, a system test plan is formulated, which verifies the functionality of an application that are described in previously gathered requirements. As depicted in figure 2**,** development then proceeds to high-level design phase, where designing of system architecture takes place. Moreover, integration test plan is also created in this phase. The ability of different components of software to work together is evaluated by an integration test. In low-level design phase, each component of software is designed. Unit test plans developed in this phase verify all components that were designed previously. After the completion of low-level design phase, actual coding begins. Once implementation is finished, development process moves to the sequential steps that lie in the right-hand side and moves in

upward direction. Thus, the final product is obtained at the successful completion of system testing.

V-shaped model is very straightforward and the extensive use of Test Driven Development (TDD) provides high rates of success. However, like in waterfall model, requirements need to be well-defined and understood. Therefore, the model is only suitable for small projects.

Spiral

Software development experts soon realized the shortcomings of waterfall model. As a consequence, many SDLC models were developed to answer the weaknesses of waterfall model, the spiral model being one of those models. On contrary to waterfall approach of software development, the newer models introduced iterative development. However, the amount of iteration varied from model to model. In addition, strict sequential development was discarded. [12, 25-26.]

In the spiral model, a team starts off with a tiny chunk of requirements. Then, they go through all the development phases with the same set of prerequisites (except maintenance and deployment). Once those requirements pass all the test cases, a new set of requirements is added and the development process proceeds in the same way as in the first set of prerequisites. However, lessons that were learned in the first cycle are considered seriously via risk analysis while moving forward. Risk analysis is a very important aspect of spiral model. The development process continues in ever-increasing “spirals” until the application is ready for deployment. [12, 25-26.]

The unfinished product that is obtained at the end of each spiral (except the final one) acts as a prototype for next iteration. The figure below illustrates the spiral SDLC model.

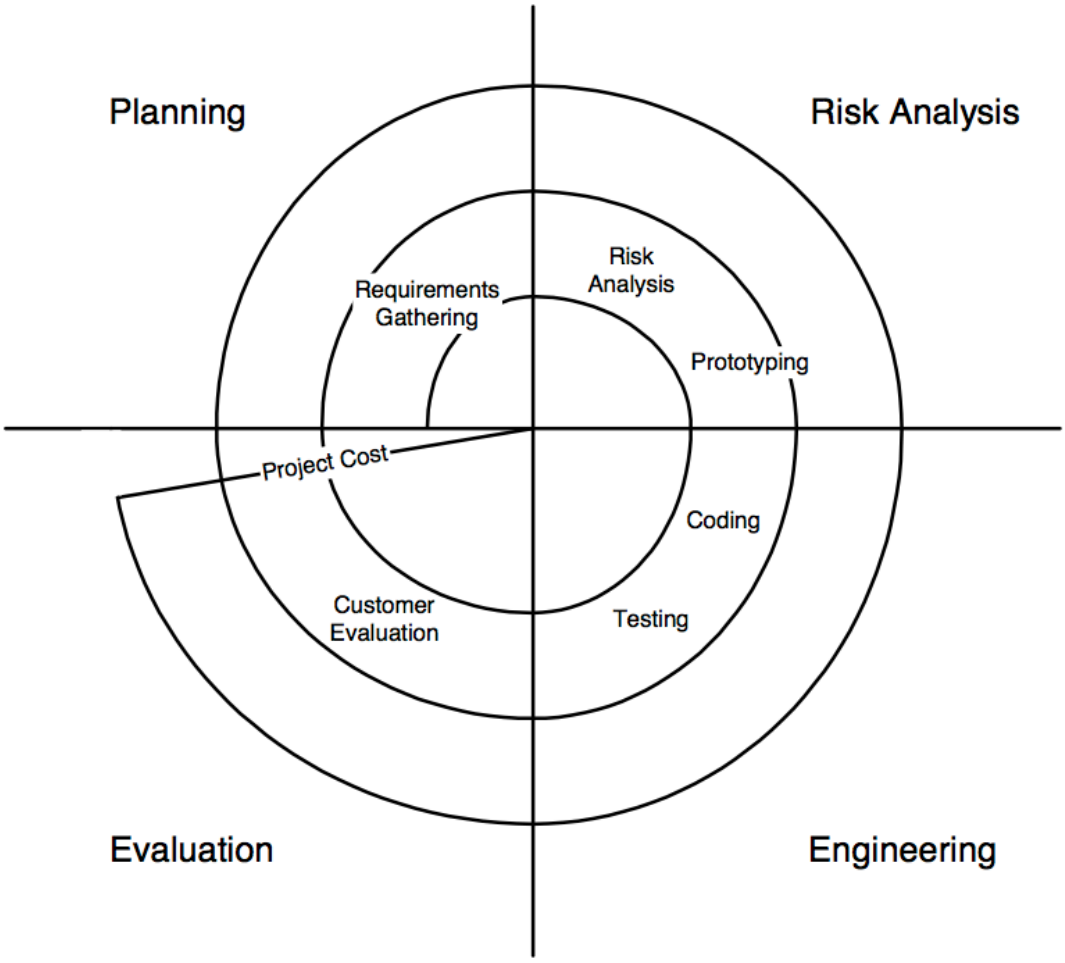

Figure 3. Spiral software development model. [6, 62]

As depicted in figure 3above, spiral model is divided into four development phases and each spiral goes along all the phases. At first, planning is done, then, designing of the system is accomplished. Later, actual implementation is carried out and testing and evaluation processes are executed in the last quarter of the development work.

The biggest edge that spiral model has over waterfall model is the ability to commence development even if the application requirements are vague and uncertain. Only the iterative process followed by spiral model provides such an ability. Furthermore, each prototype is well tested, which helps to keep the project in the right track. In addition to that, user feedback is taken for each and every prototype that guarantees the usability and efficiency of a system. On the other hand, detailed risk management sessions help

in preventing wasting the resources. Even though spiral model complies with various aspects of agile model such as incremental iterative development, it falls into the category of traditional methodologies because of its bulky and heavy development process. In addition, extensive planning is encouraged. [12, 26-27.]

Unlike waterfall, spiral model can handle projects with an element of uncertainty. For instance, online shopping application such as ebay with a great amount of obscurity can be developed using spiral model. Usually, the cost of the spiral projects is exceptionally high because project period tends to be long. Moreover, the heavy amount of documentation at intermediate phases make the management fairly complex business. In most cases, great deal of skill and expertise is required to manage the spiral projects. Also, great determination and strict obedience of principal is required to enforce this model, which frequently becomes a difficult job for developers. Hence, it is a bulky model that is best suited for big and high-risk projects.

3.2 Agile software development methodologies

On contrast to principle of traditional methodology, agile approach welcomes changes. In fact, the whole idea behind agile development is about feedback and change. No matter how simple or small the project is, change is inevitable in software development. Software can never be made perfect. Hence, spending time for heavy planning is not worth it. Why not just do it rather than think about it. In agile development, massive planning and detailed design are discouraged because priorities and requirements are always shifting during development cycles. The main idea is to obtain a decent product by iterative and incremental cycles. Different phases such as requirement gathering, designing, coding and testing are repeated over and over again. The agile principle believes in improving and developing software in small steps. Each of these steps is called as iterations. All fundamental phases of software development are executed in each cycle. The agile model advises to spend little time in early planning and more time in planning of iterations. Basically, a huge chunk of work in traditional approach is broken down into small pieces in agile model. [13, 7-11]

Today, most of the software companies follow agile software development process against traditional development methods. The term ‘agile development’ was coined in 1990s. Later, well-known figures in software development society like Kent Beck, Ward

Cunningham, Alistair Cockburn, Jim Highsmith, Ken Scwaber and others developed the term into a standard software development model with a same name. The team later wrote a manifest for agile development. They provided simple suggestions that were very easy to understand. Indeed, most of the agile thinking is merely common sense. Agile processes respond to the changing environments and requirements rather than following a process blindly. According to Williams and Cockburn, agile development is all about feedback and change. Furthermore, they describe it as the process that accepts change instead of refusing it. Agile manifesto highlights following four vital points.

1. Stakeholders should be the part of a development team and must follow the process.
2. Development must be carried out in small iterations.
3. Team decides what to do.
4. Change must be welcomed and adapted. [14, 17-18]

There are several agile methods, which are proven to be efficient. Some of them are explained below.

Extreme Programming

Extreme programming (XP) is an agile software development model that emphasizes on user feedbacks, which are provided by short iterations. Kent Beck introduced the first agile SDLC model in an attempt to overcome the challenges of linear models. [7, 99.] This model is very lightweight. Furthermore, XP provides great deal of flexibility to the development process. The risks involved in projects following extreme model are usually minimal.

Like any other agile approach, XP is an incremental development model that responds to changing requirements. Nonetheless, it is best suited for a team of two to ten people. It is to say that extreme programming is for the team that can fit into a single room. Extreme programming is designed for small teams because it relies heavily on oral communication. Likewise, pair programming is the unique aspect of XP, which makes it difficult to implement it in big teams. In extreme programming, a pair of programmers shares one machine. One of them does the coding and the other one asks questions and gives suggestion. By doing so, things that are slipped from one mind are grasped by another one. Role of observation and coding can change during the course of de-

velopment. The pair can be interpreted as the captain and his first officer of an airplane. [8, 62.]

There are certain protocols that need to be followed to be able to call yourself a XP follower. For instance, tests need to be written; there is no way out of this. Similarly, the customer needs to be involved from the very beginning of project. Customer participates in planning phase and follows the process until the end. During that period, customer continuously gives feedback, to which extreme programming responds and required changes are implemented. A team is not extreme if the customer is excluded in development process: end of discussion. A team does not get to choose certain things if it wants to follow extreme programming. [15, 9.]

Similar to every other SDLC model, first phase in XP is the requirements gathering phase. Nevertheless, as discussed above, not only technical people but also customer is deeply engaged in planning. As a point of fact, customer provides a huge piece of information on how to proceed forward. Customer set priorities and deadlines of the project. Likewise, the customer also provides requirements of the system. [15, 9.]

Simplicity of design is the beauty of extreme programming. Complex designs are totally discarded by XP because future is uncertain anyway. Hence, designing is carried out for smallest component of the software. After completion of that design, it is tested to look for a possibility to improvement. Refactoring is the keyword used for such checking. Refactoring is limited to the code of an individual developer. Coding standard needs to be set if refactoring is to be carried out in an entire system. [16, 24.]

Rapid feedback is crucial aspect of XP. In XP, implementation and test cases are written simultaneously. All these tests need to pass before the implementation of new codes. Once coding is completed, entire test suit is executed; again all the tests must run successfully. Hence, an application is always running successfully. Furthermore, testing results are gathered in a small amount of time. The latest delivery of tests result would be a few days at most. In normal case, test results start to appear within hours. Small pieces of software are tested as they get written. Therefore, problems are immediately realized and rectified. Such approach leads to the frequent delivery of bug free system to the customer. When the customer gets his hand on a working piece of software he can evaluate it and give instructions about next feature that needs to be added. [16, 25-26.]

In a nutshell, extreme programming is a very powerful agile Software development model. However, it is not suitable for all kind of projects. A project with a big team and missing customer cannot make use of XP.

Feature Driven Development (FDD)

Jeff De Luca and Peter Coad introduced feature driven development in 1997. Jeff got the idea when deadline was slipping for one of his own project in Singapore. FDD is a pure agile method that provides high adaptability towards changing environment. [17, 3.]

Feature driven development focuses on quality at all development phases. It generates tangible result in every cycle. Moreover, it is well-known for delivering deliverables frequently. Likewise, progress-monitoring mechanism is accurate and reliable. In FDD, features are prioritized. Furthermore, cost involved in each feature is listed along with its priority definition. Feature driven development involves five different iterative processes. At first, overall feature of an application is listed. In second phase, those features are prioritized and defined extensively. In next process, prioritized features are picked up for iteration and the plan is laid for the execution of that cycle. In fourth process, such hands picked features designed. Finally, they are implemented in code level. There are short cycles in FDD. In addition, specific timeframe is allocated for each cycle. The cycle is terminated when time allocated is over. After that, the most recent version of the software is tested. After testing, plans for new iteration is discussed, if it happens to be the final iteration product is delivered. [18, 26-27.]

In feature driven development, there are six key roles for people. Also, there are other people in supporting roles. First, there is a project manager, who is the boss. He is the one who provides all the required resources. In addition, he is responsible for financial management. [17,5.]

Another key person in FDD is the chief architect. Chief architect designs the entire system. It is his duty to bring all members together and run a design workshop. In design session, he collaborates with other member of the team. [17,5.]

Development manager is the one who leads the daily development work. Development manager's main job is to resolve conflicts when there is a need of authority or delegation skill. [17,5.]

Chief programmers are veteran coders who lead a small development team. They participate in requirement analysis and design session. They know the entire lifecycle of the software. They instruct class owners and help them out if they could not solve a problem. [17,5.]

Class owners are the members of a small development team. They take orders from the chief programmer and do most of the implementation, perform low-level design, coding and testing. [17,5.]

Finally, there are the domain experts who are users, business analyst, sponsors or any composition of these. They participate and monitor the project from the very beginning. Their main job is to keep the project in right track. They are the ones who provide requirements for the system. They need to be excellent in communication and presentation skills. [17,5.]

Apart from these six key roles, there can be other supporting roles like release manager who keeps track of project, so that it can be released in time. There might even be language guru who is expert in the programming language that is being used. These kinds of supporting roles can be added according to the need of project. [17,5.]

To conclude, feature driven development is the effective agile SDLC model, which uses incremental short cycle to develop software. Its main aim is to produce tangible results in timely manner.

3.3 Comparison between traditional and agile development process

Traditional methods of software development are old but yet they kept the software industry going for a long time. They were quite popular at their times. When waterfall model was introduced by Royce in 1970, it was the best thing ever that had happened to the software industry. Nonetheless, every good thing has an end. Traditional methods faded away when more efficient agile methods were developed, the first being the extreme programming introduced by Kent Beck. Agile methods only rectified the limitations of heavyweight models. As a result, both of the models follow the general software development pattern. Nevertheless, they obviously have their differences in the way they pursue the steps that are involved in that pattern. [7, 35.] The concise overview of the differences between traditional and agile methods can be seen in the table below.

	Agile Methods	Heavy Methods
Approach	Adaptive	Predictive
Success Measurement	Business Value	Conformation to plan
Project size	Small	Large
Management Style	Decentralized	Autocratic
Perspective to Change	Change Adaptability	Change Sustainability
Culture	Leadership-Collaboration	Command-Control
Documentation	Low	Heavy
Emphasis	People-Oriented	Process-Oriented
Cycles	Numerous	Limited
Domain	Unpredictable/Exploratory	Predictable
Upfront Planning	Minimal	Comprehensive
Return on Investment	Early in Project	End of Project
Team Size	Small/Creative	Large

Table 2. Summary of differences between traditional and agile methods. [7, 35]

Table 2illustrates the nature of agile and heavy methods. For instance, agile methods are adaptive and based on iteration. However, heavy methods are predictive and based on limited cycles. Furthermore, upfront planning for agile methods are minimal and is comprehensive for heavy methods. All the typical differences have been listed in table 2.

Generally, people choose development methods depending on their needs. It is obvious that both agile and heavy methods have their strengths and weaknesses. Usually, a customized method that fits best for a certain project will be chosen.

3.4 Team Development

As we know from above, software development is not an easy task. There are many variables that cannot be anticipated beforehand. At the beginning of the development process, the obvious approach to develop a solution is to look for something similar that has been made before. It is irrelevant to form a team to develop solutions, which are already available in the market. For instance, if somebody wants a blog for himself, then perhaps the best way to get this done is to use some sort of Content Management System (CMS) like WordPress or Drupal. These systems are fairly easy to use and they offer a lot more features than a solely developed solution. Furthermore, most of the CMS are available free of charge. Therefore, if something has already been made, a wise decision would be to go along with it. It saves both, time and money. [19, 17-16.]

There are lots of open source projects available in the Internet. Nevertheless, most of the open source projects tend to give a solution to generic problems. It is very likely that the problem is very peculiar, if the development has to be done for a company. [19, 9-10] A company would not invest in a solution that can be found freely. Hence, a group of competitive personnel is required to find a solution to all the difficult problems that the contracting firm has.

Moreover, a team is required to implement the customized top-notch software that can fight and survive today's competitive market. Now a day, companies are not just looking for a mere application. They demand agility and flexibility and expect it to cope with the ever-changing business needs. As an example, CMSs like WordPress and Drupal can get the site running in less than 10 minutes but that is not going to be anywhere near the expectations of the client. In order to satisfy the client, an application needs to be customized and it should address the needs of client's business. A project like that surely requires more than one person. A software development team usually consists of product owner, designers and developers. The size of the team varies according to the size of the company and the project. In small firms, teams normally have 2 to 5 members. [19, 9-10] On the contrary, a team might have hundreds of members if the project is complex. For example, there are hundreds of people working for a betterment of facebook. Basically, we can consider each and every member a part of single team. However, it is a general practice that a large team is divided into small teams.

Personally, I have not seen any developer who has not worked in a team. I myself have always worked in team whenever I have participated in an application development projects. It would be unwise not to take a team approach while building an IT (Information Technology) solution.

3.5 Why methodologies at all

A software engineering principle is a standard process that software engineers follow during the development of an application. Some of the examples of such principles would be Scrum, Waterfall model, Code and fix, Code reuse and Refactoring. Software engineering methods set the clear track for entire development process. In other words, software engineering principles tell developers “how work is done”. Following a process has some major benefits, which are explained below.

Introducing new member to a team

It is very common to bring a new member to a team while developing applications. The need of a new member may arise due to change in requirements. Sometimes, it is planned beforehand to introduce a fresh member with particular skills set in specific time. The specific time refers to a point when some tasks are completed or some resources are available for disposal. As an example, companies may decide to bring in coders when designers complete the wireframes and all the software specifications are written. Moreover, it is very common to hire people for certain duration of time in software development business. A designer is a very good example. Once the designer accomplishes all the graphical work, he rarely follows the development process.

Companies tend to employee people for the shortest time possible because they want to save costs. For small companies, this is crucial because they posses limited fund. On the other hand, big companies hire all sorts of people on permanent contract because they have many projects going on.

A newcomer can grasp the idea very easily, if there is a systematic method involved. It would be a disaster, if he has to go through all the random works done by other teammates. [20, 204.] Therefore, it is absolutely essential to use software development principles to help newcomer to get hold of the project.

Substitution of people

In companies, there should always be room for the unthinkable. A team is formed with a solid goal and specific task is given to each and every member. Companies do not want to replace someone if that might jeopardize or delay the project. An employee is an asset that the company has invested in. Firms often asks question like, “How long do you plan on working here?” before hiring someone because they want to be absolutely sure that the new employee will work for a reasonable amount of time. In many competitive jobs, a minimum commitment of 5 years is required.

Things always do not work as planned. There is always something missing or not working properly. Likewise, the unimaginable can happen in a company. [20, 204.] Thus, company needs to stay prepared. The methods used in company can make a big difference when problems arise. The newcomer can be useful in very short duration of time if the proper methodology has been used. An application developed in distributed environment benefits the most from this aspect of methodologies. [20, 204.]

Specifies responsibilities

In a company, there is a hierarchy and people with different skills. One of the key advantages of using methodologies is that it delineates responsibilities. An employee knows what he/she need to deliver. [20, 204.]

In today’s world, it is very usual to find people with multiple talents. Talent is definitely a good thing but it might cause friction when work needs to be done in a group. As an example, a programmer can be a good designer or he/she might have a good sense of colors. However, most of the software companies appoint designers separately. If tasks have not been divided properly, problems will arise. If the coder is also a proficient designer, he/she would most likely intervene if the design did not meet his expectations. Web developers are a good example of such combination of skills.

Proper implementation of methodologies can address such problems. A methodology not only tells what one should do but also clearly states what a person should not worry about. Hence, a software development principle specifies that a designer is the one who makes decisions about design of an application, not the coder; programmers implement the design via code; not the designer. [20, 204.]

Influencing the customers

It is very important that a customer gets a good impression of a company. A loyal customer is the biggest asset of any business. Let's assume there are two companies that are bidding to work with me. The first one keeps track of time and all the work that has been done and has to be done. Moreover, it documents everything very clearly following a precise method. On the other hand, the second bidder does not follow any process; rather all the team members sit together and work independently. They share notes to share ideas and status of project. They tackle the problem when they face it. The key element in succession of the project would be the commitment of a team.

In a situation like this, I would definitely choose the first bidder because well documented and planned working process will make me feel safer. I would not trust a group of personnel who tries to convince me by saying, "We are all responsible individuals and we will deliver what we commit". It is just not good enough. Hence, use of methodology can give an edge in getting a work contract. [20, 205.]

Monitoring progress

One of the key features of most development procedures is that they provide a possibility to track the progress. Tracking progress during application development is very important because it helps to stay in schedule. Furthermore, it motivates the developers. Visible progress really adds up an extra energy to the whole team. In the same way, it can be a source of frustration. In the development phase, there are times when members spend considerable amount of time working but there are no results. This often happens when a team has to do a lot of background studies. [20, 205.]

In software industry, it is very common that a customer is involved in development process. In fact, it is highly recommended by the contracting software company. Once the building process has begun, clients demand visible progress. They do not understand any technical excuses. Thus, something needs to be produced in daily basis; this is one of the key elements that I have learned in my career. An agile development method such as scrum really pushes developers to produce something everyday. In scrum, a member of the team needs to explain the status of his work in scrum meetings, which are held everyday. [20, 205.]

Many companies make use of burn down charts to monitor their advance. A burn down chart is a graphical representation of remaining work against time. In simple words, a burn down chart is a run chart of work left. The name 'burn down' implies that the amount of work is decreased as the time increases. It gives a clear picture of project status at one glance. In many instances, remaining tasks represent the vertical axis and time is along the horizontal axis. A burn down chart is a very flexible graph because it does not require any specific units. A graph producer can make use of any convenient measurement unit for both time and work. It is also acceptable to use a unit of one's own if that makes sense. Scrum makes an extensive use of burn down charts. Nevertheless, it can be implemented in all kind of projects, which have measurable work and time. It is good to set the date of product launch because it makes the employee to work hard if something has not been done. The figure below is the very good example of a burn down chart.

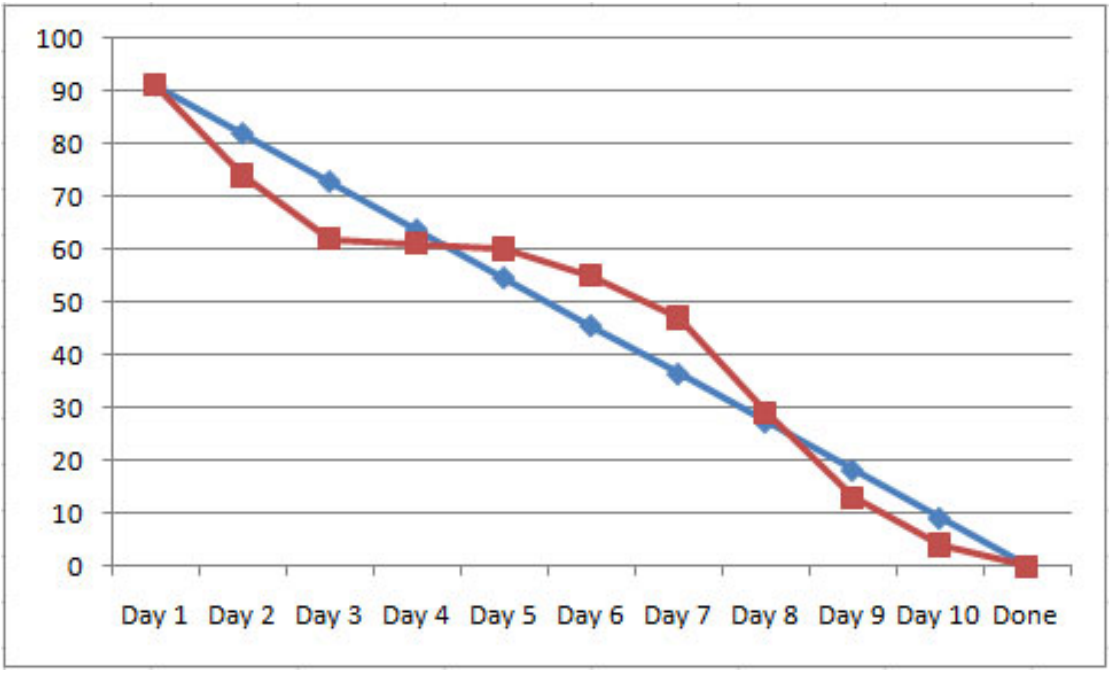

Figure 4. A typical example of a burndown chart. [21]

Figure 4 demonstrates that ninety tasks have been completed in duration of about eleven days. It also shows that more than fifty percent of total tasks have been accomplished in last four days, which is a usual case in software development. Most of the work in application development is done in later hours of the project which raises the stress level substantially.

Educating employees

Once the company chooses a standard methodology, courses can be designed to educate employees about techniques and skills that work best with that particular method. It makes the work much more efficient. As an example, a company can train an employee to be expert in UML (Unified Modeling Language) drawing or in using any particular tools. Based on their responsibilities, workers can be sent to training. [20, 205.]

By doing so companies will turn their employees into experts, which will definitely yield more profit. Moreover, companies win loyalty from workers because they are given extra training for free.

4 Scrum

Scrum is an iterative and incremental framework that facilitates a development of software product. Actually, scrum can be used to manage any kind of work not just software development. [22, 3.] The term “scrum” has been derived from the sport of rugby where it is the means of resuming game when minor foul occurs [23, 132]. In rugby, Scrum is an important event to employ strategy and teamwork. Ken Schwaber and Jeff Sutherland developed scrum in 1990s. They define it as a framework rather than a method within which techniques and processes can be employed. [22, 3.] In this paper, scrum has been explained extensively because it is the method that was employed during the practical phase of this thesis.

Like any other agile method, scrum provides great deal of flexibility towards changing needs of an application. The figure below gives the glimpse of working process in scrum.

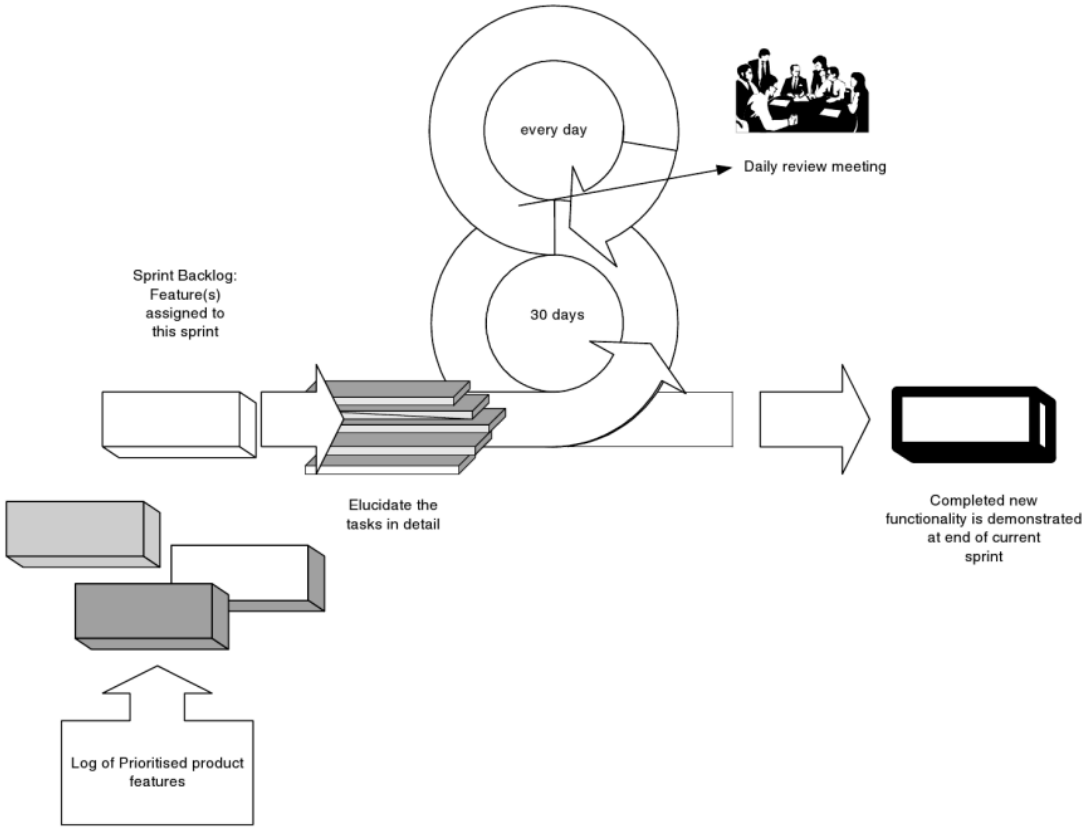

Figure 5. Scrum Framework. [18, 26]

Figure 5 shows the events and process involved in scrum. At first, product backlog is created. Then, sprint backlog items are selected from the product backlog items. Such

items are fine-grained and implemented in a sprint. There are daily scrum meetings during 30 days of sprint. Finally, usable or even releasable increment is delivered at the end of sprint.

Those who have practiced scrum consider it very easy to understand but extremely difficult to master. Similar to extreme programming, people need to obey certain rules to consider themselves a scrum follower. Special scrum rules, events and artifacts are described below.

4.1 The Team

To practice scrum, there needs to be a team. Such team is called as ‘Scrum Team’. The scrum team comprises of scrum master, product owner and the development team. [22, 5.] Each and every member of a team has a special role. Usually, scrum team is cross-functional in nature. Moreover, it is optimized to increase productivity and creativity.

In addition, scrum teams are self-organizing in nature. Self-organization is the team practice, in which there is no command and control. Furthermore, the team manages themselves. Self-organizing team has the power to make decisions. Moreover, they pick their own task against the conventional approach where leaders used to do it for their employee. Likewise, these teams communicate heavily among themselves and they are continuously improving their skills. A member of self-organizing team understands the project requirements and is committed to the team instead of a boss. [24.]

Self-organizing teams still require mentoring and they are not entirely free from the grasp of management. Leaders create the product backlog and they recruit team members. In addition, management takes actions if self-organizing team starts to go against its very principles. The only thing self-organizing theory denies is the absolute ‘command and control’ over team. As an example, Pekka is a senior developer who is dominating and making all the decisions of self-organizing team and nobody stands up to him because he is more experienced. [25] In this case, team members cannot question freely, which deters the ability of improvement. Additionally, they cannot put their ideas forward, which is the key reason why self-organizing teams came into existence. In a situation like this, management has to intervene. However, intervention might

come in an indirect form. For instance, management can bring in someone who can question Pekka's decisions. Direct interference in a case like this does not yield beneficial results because team starts to fear and wait for leader's decisions after such incidents.[25.] Hence, management has to play a role of strategist while managing self-organizing teams.

Self-organization is the key element in agile software development. The agile manifesto encourages self-organizing team. [25.] However, team members must have some qualities to be a self-organizing team player. They need to be competent and highly skilled. Furthermore, they are required to be good team players. Likewise, they should have an ability to trust and respect others. [24.]

Since scrum teams are self-organizing, they usually make decisions and manage their work. Members of a scrum team are described below.

Product Owner

A product owner is accountable for the performance of the team. He or she is the one who provides all the requirements of a system. In real life, product owner is the stakeholder. Nonetheless, according to scrum principle, this is not obligatory. Therefore, the project owner can be any person with good management skills. Product owner sets deadline and pushes development team to complete their task within given time. Setting priority of backlog is also part of product owner's responsibility. Moreover, it is his/her job to make the development team understand the product backlog properly. [22, 5.]

Development Team

Development team is a group of professional software developers. They take care of technical development. They create and implement a test design. Finally, they test such developed product. It is their responsibility to deliver well-tested product at the end of every sprint. However, the product does not need to be the final version unless it is a final sprint. The size of the development team varies according to the project's demand. Nonetheless, a team should consist at least of three members; otherwise productivity of a team might be compromised because of the skill constraints. On the

other hand, it should not exceed nine members because coordination might be difficult. [22, 6.]

Scrum Master

Finally, there is scrum master who is the de facto leader of scrum team. It is his/her responsibility to make sure all the scrum principles are being followed. In addition, he/she organizes daily scrum meetings and receives status report from each developer. Scrum master also divides work to developers. It is a very common practice to assign the most experienced member of the development team to a role of scrum master. [22, 6-7.]

4.2 Events to take place

Scrum framework strictly defines several events that need to occur in a development process. These events have been designed to increase productivity and adaptability. Furthermore, it decreases communication gap, which is the biggest challenge of team development. [22, 7.]

Sprint

Sprint is a soul of scrum. Schwaber and Sutherland define it as a time-box within which a decent amount of product increment is produced. The duration of sprints may vary from project to project, however, it is uniform throughout the same project. The time duration of sprints is usually less than a month but more than a week. Next sprint starts immediately after the retrospective session of the previous one. A sprint can be concluded if all the goals are achieved before the sprint deadline. Only the project owner has the power to terminate sprints before the end of sprint time-box. However, it should be kept in mind that scrum master and development team have great influence over him/her to make this decision. [22, 8.]

Sprint Planning

Before the commencement of each sprint, there is a planning session. Sprint is planned by the collaborative effort of each member. Sprint planning meeting is also

time-boxed. For a sprint of one month, the developers of scrum framework have suggested the duration of eight hours meeting. Nevertheless, time-box may vary depending on the length of sprint. In sprint planning, tasks are divided and sprint backlogs are picked from the product backlog. [22, 9.]

Daily Scrum

Another important event in scrum is the daily scrum meetings. It is also a time-boxed event. Schwaber and Sutherland have advised duration of 15 minutes. The main purpose of scrum meeting is to revise the work that had been done in last 24 hours and to predict the work that can be accomplished in next 24 hours. [22, 10-11.]

Sprint Review

At the end of sprint, there is a sprint review session. In this session, overall increment is inspected and adaptation is made if required. Scrum team and stakeholders take part in sprint review session. [22, 11.]

Sprint Retrospective

Finally, there is a sprint retrospective session, which occurs before the beginning of a new sprint. Its main purpose is to enable team members to talk about the wrong doings of previous sprint and lay a plan to avoid them in next sprint. It is also a time-boxed event. For a sprint of one month, duration of three hours is allocated. The time duration is decreased proportionally if sprint time is shorter. [22, 11-12.]

4.3 Artifacts of Scrum

Scrum artifacts mean the work that is carried out to facilitate transparency in development work. Moreover, they were designed to optimize adaptation and inspection. [22, 12.] Scrum artifacts have been described below.

Product Backlog

Product backlog is a list of features that are going to be needed in a system. It is an ordered list, which acts as a single source for changes in an application. The responsibility of producing product backlog lies with the product owner. It is his or her job to prioritize and make it available to development team. [22, 12.]

Product backlog is a dynamic artifact. It is never complete and it is continuously evolving. At the early stage of development, it only consists of very first requirements that were understood. The product backlog exists as long as the corresponding product is in existence. [22, 12.]

Features listed in the product backlog have attributes, namely description, order and estimate. Backlog items are ordered on the basis of risk, necessity, priority and value. As a consequence, there are low-ordered and top-ordered backlog items. Top-ordered items have detailed description and are precise estimates. On the contrary, low-ordered backlog items are poorly described and estimated. Backlog items that are to be implemented in next sprint are fine-grained before the beginning of that sprint. [22, 13.]

The development team selects backlog items so that they can be accomplished in deadline. It is development team's job to decide all these estimates. [22, 13.]

Monitoring Project Progress

Monitoring progress is a very important scrum artifact. In scrum, remaining work at any point of time can be calculated. Product owner is always tracking the total remaining work. Such information is made transparent and all stakeholders know about it. To monitor progress, projective practices like burndown and burnup are used. [22, 13-14.]

Sprint Backlog

Sprint backlog is a list of items that are chosen to be crucial, they form product backlog items which should be implemented in a sprint. The development team is responsible for selecting sprint backlog items. It is a definition of work that is going to be added in a sprint. Sprint backlog is also modified as sprint progresses. Only the development team

has a right to perform such modification. As items in sprint backlogs are added, deleted and completed, development team updates the remaining amount of work. Sprint backlog provides a clear picture of work that is going to be completed in a sprint. [22, 14.]

Inspecting Sprint Progress

Sprint progress can be summed at any given point of time. As sprint backlog solely belongs to development team, it is their responsibility to track sprint progress. Tracking of remaining work is performed at daily scrum meetings. [22, 14.]

Increment

Increment is the total amount of work that has been completed in a sprint and all the sprints that occurred previously. It is a sum of all the accomplished backlog items. However, increment must be usable and well tested. It is obligatory that increment meets the definition of “Done”. [22, 14.]

Definition of “Done”

Definition of “Done” is a scrum term, which is used to mark a backlog item as completed. It is very important that all the team members understand the meaning of “Done”. Definition of “Done” varies from project to project. There are no specific guidelines for defining “Done”; scrum team defines it according to the needs of the project. [22, 15.]

5 Distributed approach to software development

The way that software engineers develop an application has shifted drastically in the last two decades. Technological progress has made the impossible possible. Rapid growth of Internet has carved a path to the dispersed software development of today**.**

The idea of geographically separated development came into existence when physicist and researcher Jack Nilles first used the term ‘telecommuting’ and ‘telework’ in 1973 for a work environment that does not require any commutability from employees. The teleworker works at home or at some remote place to complete whatever job he is responsible for. Jack Nilles is also known as “Father of telecommuting and telework”. Companies did not have enough technological capabilities to use that idea then. However, explosive growth of global market and availability of economically cheap workforce on the other parts of the world has caused a rapid development of tools and technologies that made the scattered developments possible. Today, thousands of individual work with other individuals in various part of the world to build software but face-to-face meeting between them may never occur. [26, 1-2.]

5.1 What is Distributed Software Development

The software development process that involves at least one geographically scattered member is known as distributed software development. In this approach, all the team members are not collocated in the same place. Therefore, a team member is not able to have in-person communication with all the members of a group. However, part of a

team can be together and face-to-face meeting among them is possible. Moreover, there can be separate teams in different geographical locations with a specific task but contributing to an outcome of the same project. The figure below illustrates the concept of dispersed software development. [27,122]

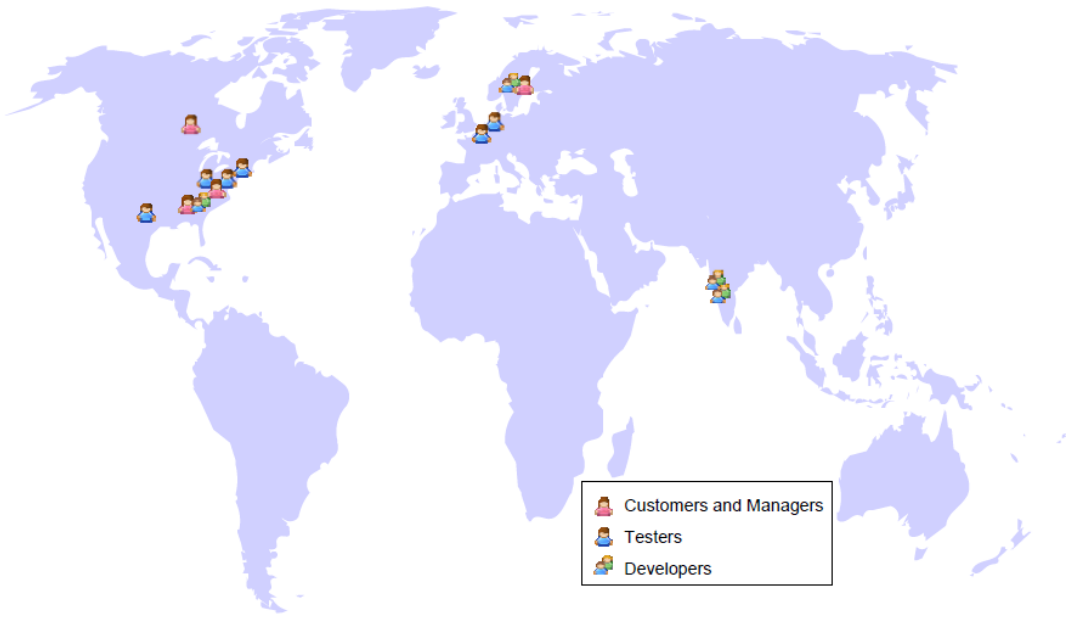

Figure 6. Illustration of distributed software development. [27, 93]

Figure 6 provides an overview of scattered software development. It is important to notice that individuals are not just distributed geographically but functionally also. Developers are in Asia and North America. However, testing of an application is being carried out in Europe and North America only. Moreover, there are not any managers and clients in Asia.

It is a well-known fact that without technological achievements scattered development is just a myth. Beside technological progress, there are other scenarios that pushed companies to put trust and fortune into overseas workstations. Companies face problems like deficiency of skilled manpower and resources. Moreover, production cost of software in western countries is way higher than in Asian or South American countries. These are the major reasons why distributed approach is going viral. Software houses are desperate to bring out new applications into the market but it is absolutely essential for them to deal with these problems first. Hence, huge firms started to shift their development approach from local to global environment. The main purpose of dispersed development is to reduce the production cost and get access to skilled resources.

5.2 Challenges in Distributed approach

In distributed development, developers work in a unique environment. They need to cope with a difference in culture, time zone and site of work. These circumstances create unexpected problems. Usually, issues in dispersed environment are of different nature such as cultural and social. Obviously, these issues affect the development process and the quality of the software. Some of the most crucial ones are discussed in this section.

Communication

Since, team members are not collocated in same place, communication among developers is the major challenge in distributed development. For development of good software, engineers need to spend considerable amount of time in giving, taking and understanding information among members. Communication pattern directly affects the quality of software. Hence, standard protocol for communication is advised in distributed development. A communication without rule can result into late responses and misunderstandings. To deal with this issue, a software development method must be backed up with proper communication tool. A good communication tool provides user-friendly interface and smooth connection. Many scholars recommend this approach to eliminate communication gap without affecting the outcome of the project. [28, 29.]

Likewise, difference in time zone can create problem to set up a time for meetings. Both parties should discuss and agree on a same time before commencing a project. Common mistake in agreeing time is not to mention a time zone but only a time. As an example, if two developers working in Finland and Switzerland separately agree on time 9 o'clock in the morning, they must specify if it is a Finnish or Swiss local time. Prior agreement can easily sort out this issue. [28, 31]

Similarly, cultural difference is another important aspect in communication among team members. Different culture means different meaning of gestures and terminologies. Such difference can cause serious mistakes in translation and understanding of information. Hence, standard guidelines must be used to process translations. [28, 30-31.]

Additionally, software projects can have sensitive information, which are absolutely forbidden for public or rivals. Thus, security of communications must be considered. Proper training and guidance must be provided if it is felt as required.

Team Spirit

Dispersed team lack team spirit in many occasions. It is because of the isolation from rest of the team members. One of the characters of human being is an ability to connect with people. They create bonds by talking and sharing thoughts. Hence, level of team spirit is really high in collocated teams. On the other hand, distributed teams do not have the luxury of making jokes with their teammates, which critically damage spirit of the team. Lack of team spirit may result into lack of trust among members. Furthermore, team spirit is really important because it directly affects the productivity of a member. Low productivity means ill outcome of the project. To cope with this problem, managers can set up an annual in-person meetings or recreational tour to some place. These kinds of activities not just provide chance for bonding but also bust the level of trust among each other. [29, 7.]

Status Awareness

To make good software, developers must know the situation of the project at all time. Feelings of indifference and limited socialization in dispersed development can easily create confusion among developers. Moreover, changing requirements and priorities cause distributed teams to lose the track of work. They often do not know about the working progress of another developer. As a result, right information cannot be found in time. Hence, some sort of visualization tool is necessary to track the progress of work. Many experts suggest a tool with notification system. [29, 6.]

Merging of Source Code

Once the code is written, it needs to be merged with the codes written by other developers. Complexity of collaboration and sharing of code increases, as the developers do not work in same place. Hence, source code control becomes critical. Most of the companies have version control system that takes care of this issue. However, version control system must have some features to be applicable for distributed teams. It needs to

work in a cloud with Internet access. Moreover, data needs to be secured while it is transit or in remote server. [29, 6.]

Knowledge Sharing

Once an application is complete, team member must save their knowledge of project for maintenance or further development in the future. The term knowledge refers to experiences and methods applied by the developer. It can be really helpful if new developer starts to work in a middle of the project. Accumulation of knowledge can save great deal of time and money. In addition, it prevents redoing of work. Experts advise to use web-based repository for getting rid of this issue. Projects can be learned very quickly with the help of such knowledge center. [29, 6.]

Risk Management

Due to addition of new risk factors such as lack of coordination, collaboration, communication and sharing, risk management is very important in distributed development. Bugs are very likely to occur because of the added constraints. Hence, risk management activities must be increased to minimize software defects. Managers can specify responsibilities and guidelines to control teams. Such controlled actions help in early detection of problem. Moreover, some experts suggested an idea of using web-based forms and templates for prevention of synchronous activities. [29, 8.]

5.3 Scrum in Distributed Development

It is a long-standing tenet that distributed development compromise the quality of software. However, more than 50% of the total software projects worldwide are distributed in some way. Researchers had been trying to adapt agile methods to increase productivity and efficiency of these projects. They have succeeded in large extent. Today, most of the distributed projects use agile methods without harming the quality of software.

Before applying agility in distributed environment, it is very essential to get oneself acquainted with agile practices in collocated context. There are not many differences in the process itself. However, scattered environment posses some peculiarities. Thus,

standard method needs to be modified to address those unique characters. In reality, it is just a process of removing or replacing the unrequired aspects of standard software development method without tweaking any core principles.

In this dissertation, integration of scrum into the distributed environment is discussed because scrum was applied as a development method during the practical phase of this paper. To scattered teams, scrum can be applied in several different ways. Jeff Sutherland and Guido Schoonheim proposed three different ways of applying scrum to distributed teams.

Isolated Scrums

In isolated scrums, there are independent scrum teams in each and every location. The work done by one team is independent of another one. There is no need of collaboration whatsoever. Since teams are totally separated from each other, scrum can be implemented as it was prescribed originally. [30, 12.]

Isolated model is best suited for the cross-functional scrum teams. As an example, if the project has development, test and architecture team at China, Finland and United States, isolated scrum would be the best choice because each team has all the expertise it requires and development can progress well, even without the collaboration. On the other hand, success breed by communication has no comparison. [30, 12.] Therefore, a team must come up with a way to compensate that missing part.

Distributed Scrum of Scrums

Similar to isolated scrum model, distributed scrum of scrums model have cross-functional scrum team at different geographical location. However, unlike isolated model, these teams are not independent. In scrum of scrums, scattered teams collaborate with each other. In other words, they act as a big scrum team, which gave the name ‘scrum of scrums’ to this model. Coordination in geographically distributed team is a challenging task. Therefore, a representative is appointed to each team. [30, 12-13.]

Totally Integrated Scrums

Totally integrated scrum is the third model that is use to manage distributed scrum teams. In totally integrated model, all the scrum team members are scattered in different locations. It might be that all the testers are in one place and developers are in another place. It might even be possible that each member are in different location and work remotely. However, all of these scrum teams are working towards the common goal. Furthermore, they are required to present an integration-tested single system at the closing of each sprint. International Business Machines (IBM) is the well-known user of totally integrated scrum. [30, 13-14.]

6 Dooxe Oy

Dooxe Oy is a service-based company that was commenced by several energetic entrepreneurs, Teemu Piirainen being the founding managing director. It is a very small company with a workforce of six people. Among six, Teemu is the only full-time worker. He handles all the management and administration of both the firm and online services that the company operates. In addition, Teemu was the project manager and product owner of this project. Jan Diener-Rodriguez is the only software developer in the Dooxe team, rest of the team members are mainly marketing and media relation specialists. [31.]

Despite of being a small firm, Dooxe uses state of the art principles for development and enhancement of its service. The lead developer Jan lives in Switzerland and is responsible for technical management. He is also in charge of maintenance and timely integration of applications. He, along with his friend developed the whole Dooxe framework.

On the other hand, Hung Ho Ngoc and I were temporary coders hired by Dooxe to develop some components of the software. Our roles can be classified as junior developers in this project. We did not have to go to dooxe office unless there was a meeting. We were to work and deliver code from our homes. Since member of Dooxe team contributes from different places, it practices distributed approach for both management and technical development.

Dooxe uses PHP as a major programming language. Additionally, JavaScript has been used extensively in client side. The service utilizes MYSQL for data storage. Moreover, zend framework has been applied as a foundation for the entire service.

6.1 The Application

Dooxe oy operates two online services namely ‘dooxe’ and ‘korjausurakka’. Customers can post any kind of job in dooxe whereas korjausurakka is strictly developed for a job relating to construction. Basically, both of these services are reverse auction platform [31]. The idea of developing a reverse auction service gave a birth to dooxe. Later,

need of separate service for construction work was felt because numbers of such jobs were overwhelming. Thus, korjausurakka came into existence.

It is very time consuming process to find a right contractor to do any kind of job. In addition, it is very annoying because price, quality and suitable time needs to be considered simultaneously. As a solution to this problem, dooxe and korjausurakka brings service seeker and provider together. Service seekers simply post a job in the service and contractors try to outbid each other by offering their best offer. Finally, customers choose the service provider of their preference. [31.]

6.2 Background on the practical work

Both ‘dooxe’ and ‘korjausurakka’ were in production when practical work for this paper begun. Thus, reader should be clear that this paper is based on studies that has been done while developing additional features to the already existing fully functional service. I did not create the whole service but parts of it. As mentioned earlier, lead developer of the project along with his friend built the service from scratch using zend framework.

Dooxe and korjausurakka have many things in common. They have same backend and share many codes in front end as well. Their color schemes are different but structure of the view is somewhat alike. We can consider korjausurakka as the optimized clone of dooxe. At first, the plan was to develop extra features to korjausurakka and later enhance those developments to dooxe. Korjausurakka was chosen as a priority because enhancements were being carried out for about a year and it was more important from the business perspective. On the other hand, improvements of dooxe were halted from long time. At that time, everybody assumed that enhancement of dooxe would not be difficult because korjausurakka and dooxe have lot in common. Thus, dooxe was planned to be integrated in later part of the project.

As planned, development team started developing additional features to korjausurakka. Later, when all the works were complete, integration of dooxe was commenced. Nevertheless, it was a nightmare. We allocated very little time for integration and our assumption was totally wrong. Although dooxe shared files and codes with korjausurakka, complexity of the service demanded lot more time than we expected. Therefore, en-

hancements of dooxe were left for senior developer. Hence, this thesis is solely based on the work delivered to korjausurakka.

6.3 Tools and technologies used

Tools and technologies that were used during practical phase of this paper will be discussed in this section. Nevertheless, irrelevant tools to this study such as text and photo editors have been ruled out.

As referenced previously, Dooxe oy is a small firm but it uses all modern approaches towards software development. It has applied framework as a foundation to korjausurakka, which is a newest way of developing web applications or any other kind of software. In addition, bug-tracking system is in place and used very effectively. Furthermore, availability of version control system provides sense of relief to coders because things can be rolled-back easily if something goes wrong. Details of tools and technologies applied during practical phase of this paper are described below.

Zend Framework

In software engineering, framework is a collection of libraries that provides reusable codes to developers. Frameworks provide basic structure of the application. Later, a developer can modify that basic structure to a software or service that best suites him. Moreover, developers can customize standard frameworks by adding their own libraries. The idea behind frameworks is that a developer does not have to build software from scratch. Reusable libraries are the greatest power of frameworks. In addition, frameworks provide amazing methods. It would take quite a while for developers to build such functions. Furthermore, frameworks provide libraries for securing applications, which are very convincing.

Likewise, frameworks help in keeping the project well structured. It prevents project from being disorganized because most of the frameworks follow software architecture pattern called Model-view-controller (MVC). Massive use of frameworks gave birth to model-view-controller. Model-view-controller is a booming concept in today's software industry. Model-vie-controller pattern contains three basic blocks: model, view and controller. These blocks are sort of groups that keep similar kind of codes. It is the main reason why projects are clean and well structured when frameworks are used. In MVC approach, model refers to a logic and state of the application, controller is a communication channel between view and model and view is a visual representation of a state of model. The Model-view-controller pattern recommends thick model but thin control-

ler and view. Most codes are placed in model so that they can be reused in future and controller or view does not need to take any unnecessary burden. The theory behind Model-view controller pattern has been exhibited in the figure below.

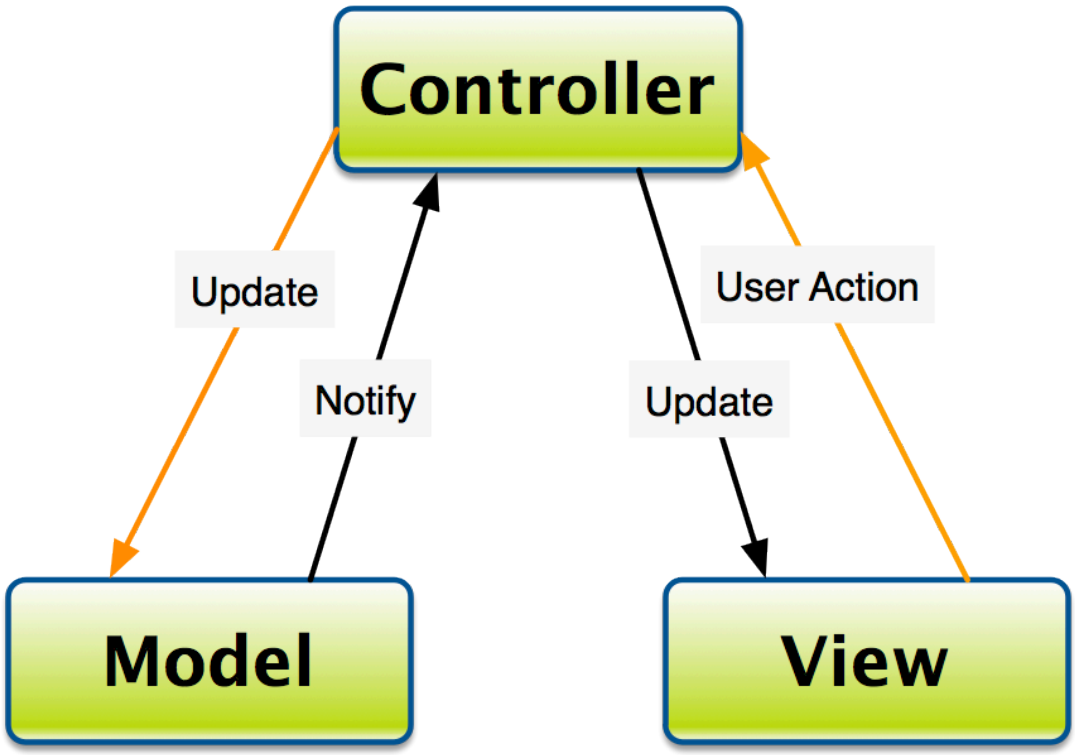

Figure 7. MVC pattern. [32]

As depicted in the figure 7 above, the controller receives instruction or input from the view, which is caused by user action. Then, controller sends those instructions to model for processing. During this process, controller calls method and object from model. After that, model processes that user action and updates its state. Change of state is notified to the controller. Then, controller updates view with the new state of model. Updating new state of model means feeding changed object and value to view. Thus, view is changed and user gets the result for his action. One of the important aspects of model-view-controller is that it prevents redundancy of code. As an example, if username needs to be displayed in four different pages, traditional web-development approach would require same code to be written in four different places. In MVC pattern, we can write one function to fetch the username from database and store that function in our model. Later, controller can call that function from model whenever it is required. It is a same concept as in object-oriented programming but MVC focuses on keeping model, view and controller separately. In web-development, logic classes and

databases are referred as models. Views are the visual representation of the service, thus HTML files are mentioned as views. Lastly, Controllers are the classes that interact between model and view.

On the other hand, applications created by using frameworks are bulky. Frameworks contain all kinds of libraries and methods. It is impossible for one application to use all of those incredible features. Thus, hundreds of lines of codes remain unused. They just stay in the service for no reason. Unused codes cause application to perform slowly. Removing those codes would be very difficult because frameworks are also complex software. One minor mistake can cause total failure of the application. However, developers turn a blind eye on this issue because performance power of computer is growing very cheap and frameworks give lot more than it takes.

There are many frameworks available for different purposes. Among them, zend framework is a PHP based web-development framework, which is the most popular framework for developing modern PHP applications. Furthermore, it is an open source project; hence, developers with proper skill can view and edit the source code if necessary. Zend framework applies the concept of object-oriented programming in every library. Thus, methods are highly reusable and can be overridden easily if necessary. Zend framework 2 is the latest version available at the moment. Zend framework 2 evolved from zend framework 1, which was very popular amongst PHP developers. The total numbers of download for zend framework 1 was over 15 million. [33.]

Git

Version Control System (VCS) is a tool that manages and keeps track of different versions of application or content of a file. Basically, it marks the changes in software. Such changes are saved along with timestamp and username of a changer. Furthermore, VCS saves the entire project in a remote server. Version control systems are also referred as source code manager or revision control system. However, authors and users of each system may argue the difference between three, each system was designed and developed to solve the same problem. Thus, all of them perform same tasks: manage storehouse of contents in chronological order, grant access to older versions whenever requested and keep log of all the changes. In this paper, the term version control system is used to mention any kind of source code manager.

The need of version control system is realized when computer programs started to become complex. At the early stage of software development, computer programs were very simple. They were written to perform a specific task. Hence, number of files and people involved in developing process were very small. Such projects did not need any version control software because every modification can be tracked very easily because entire system was very small. On the contrary, software is very complex in present context. Software today performs multiple tasks simultaneously. One individual cannot create such applications. There might be tens and hundreds of people working in a same project. Since all of them are working in a same project, it is very obvious that they alter content of files in regular basis. It would be a complete chaos if all of those modifications were left unmarked because one developer would not know who, when and where did another developer altered the contents. As a solution to this problem, software engineers developed version control systems.

The working mechanism of revision control system is very simple. The VCS software creates two separate repositories. One of the repositories is a local repository that is located in coder's local machine. Only respective coder has access and authority over that repository. Local repository is created when coder installs version control system in his private computer. Hence, it can also be called as private repository. Other repository is a remote storehouse that can be accessed by all developers involved in a project. Remote repository is created in service providers online platform at the beginning of a

project when coding is yet to start. Developers can login to online platform and look for the address of a repository. The mechanism of version control system has been illustrated in picture below.

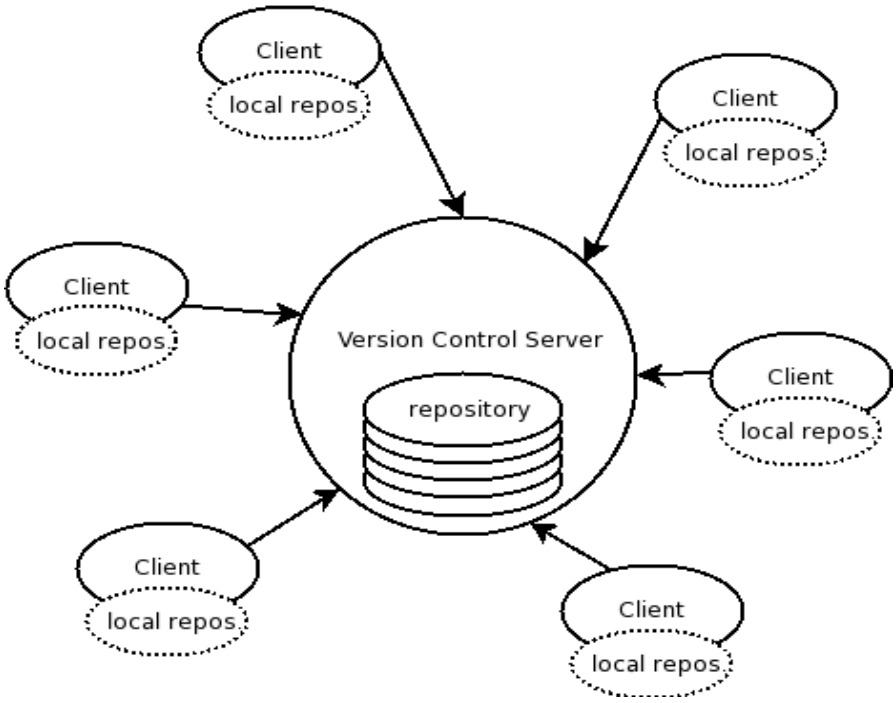

Figure 8. Overview of version control system. [34]

As exhibited above, developers have local and remote repository. Once coding starts, developers write codes in their personal computer and commit those codes to local repository. After that, they upload the local repository to a remote version control server as shown in the picture. Other developers can easily see and access those work either by simply logging into the remote server or by running a command in a command prompt.

Source code manager of today are very smart. They can save changes with different colors, which make them easily visible. Moreover, it saves different state of a file; hence it is very easy to switch back to the older version if something bad happens. For instance, 5 coders committed their codes to the remote version control server at some point of time. Nobody noticed any problem with committed codes. After some days, program started to show strange behaviors but nobody knows what is happening. In such case, it is much easier for developers to rollback to previous version and to find a problem in committed codes rather than trying to figure out a bug in entire software. Thus, source code manager tools can be of great help in finding and fixing bugs.

In addition to that, remote saving feature of version control system is astonishing. Computers and hard drives are mere machines; they can give up at any time. Unexpected crash of system causes loss of development work if they are not backed up properly. Such crash can jeopardize the entire project. Moreover, it can result into loss of job or money, which could be devastating personally also. VCS provides assuring solution to that problem by saving files in a remote location.

Another aspect of version control system is that it is absolutely unavoidable in scattered development. Fellow developer can point out the files and lines if developing environment is not distributed. Since, coders cannot have face-to-face meeting in dispersed environment, it is very difficult to ask questions about code. In such case, source code manager is the only option to follow other's work. Additionally, revision control system compiles the work of each and every developer, which would have been a terrible thing to do if there was no version control tool.

During practical phase of this paper, git was chosen for controlling versions of korjausurakka. Git is a free and open source version control system. It is designed for every kind of projects, hence can handle versions of small to large software efficiently. It is comparatively very easy to use and performance is very fast and convenient. Git works in the same way as any other version control system: saving and merging of codes. However, it possess additional features like multiple workflows and cheap local branching, hence surpasses other VCS like Subversion, Perforce and ClearCase. [35.] To be able to use git, one should install it in his local machine first. Basic pulling and pushing of code in git via command prompt has been demonstrated in the figures below.

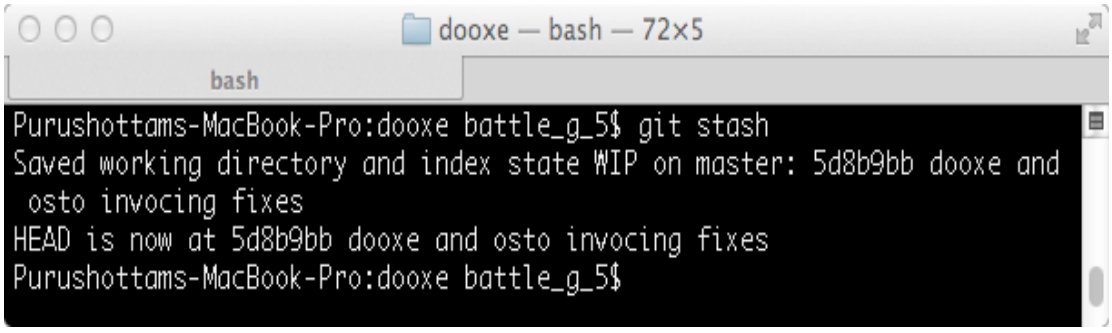

Figure 9. Stashing of changes in buffer.

Firstly, as exhibited in figure above, `git stash` command is used to store new local version of an application in buffer.

In second phase, `git pull` command downloads the latest version of software from the remote server as exemplified in figure 10.

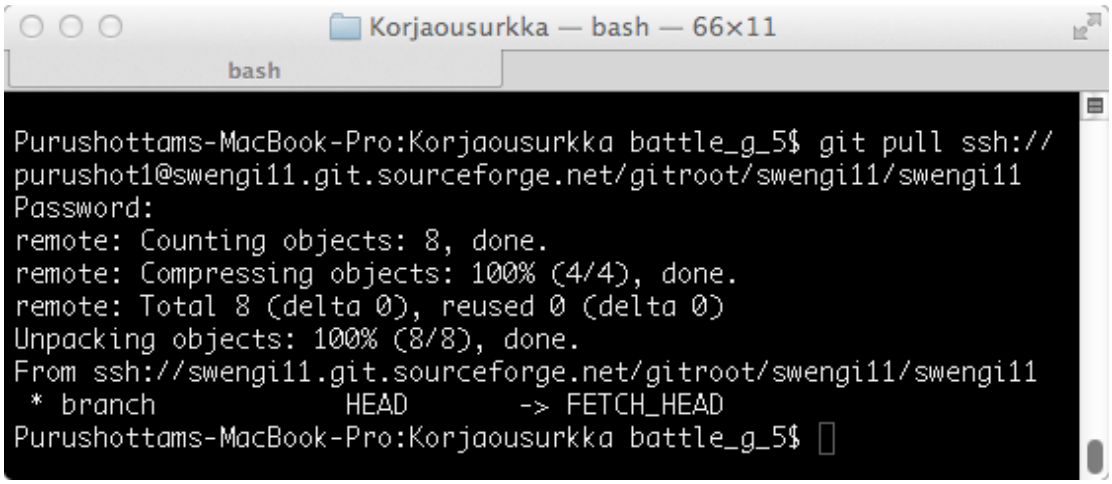

Figure 10. Pulling latest release from remote location.

In figure above, `git pull` fetches the latest version of an application from remote server. After pulling most recent version, developer needs to merge the local recent version with recently pulled version as illustrated in figure 11.

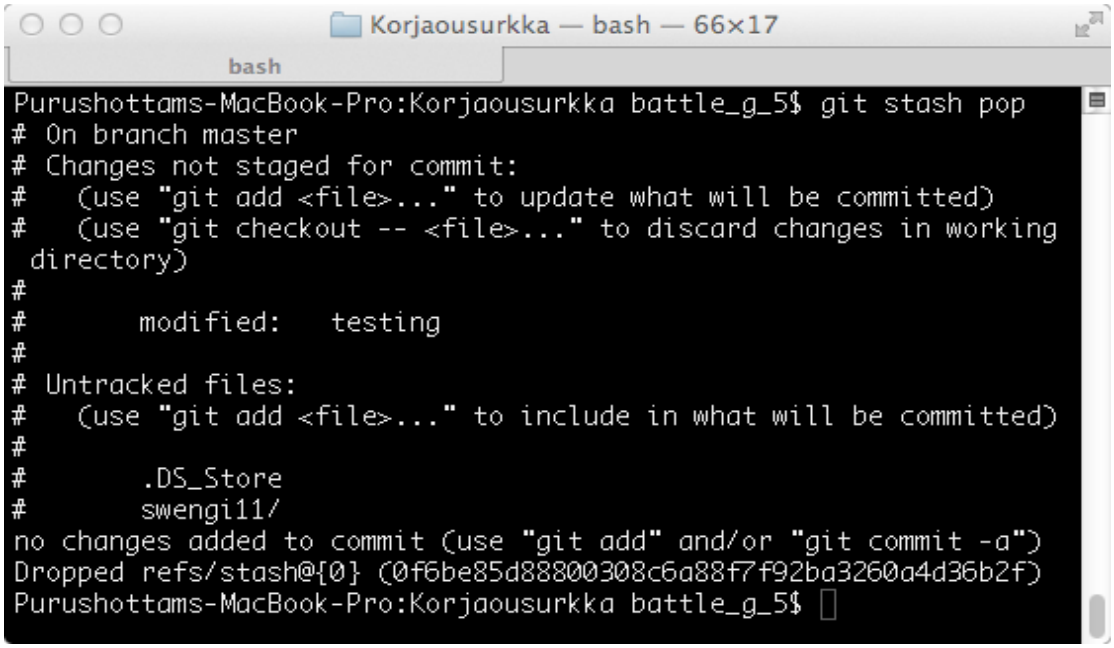

Figure 11. Merging remote and local versions.

Figure 11 illustrates the merging of local and most recent version of software. For merging stashed changes, `git stash pop` command is used. Then, merged version

needs to be committed to local repository by using `git commit` command. `Git commit` takes an obligatory parameter `-m`, which refers to a commit message.

Commit message is essential for committing a code to repository. Commit to local repository has been visualized below.

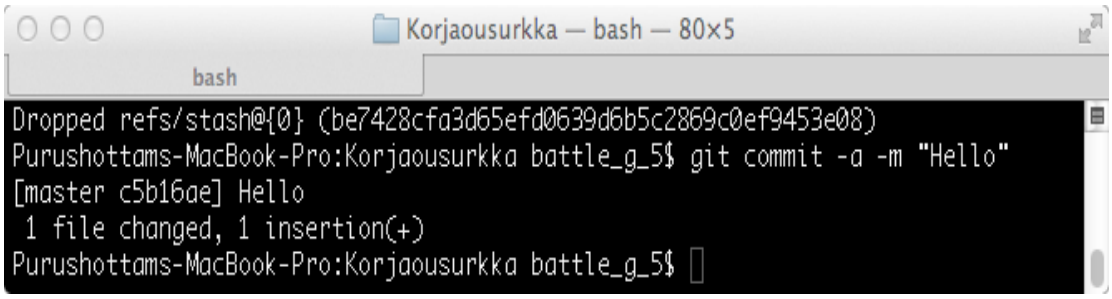

Figure 12. Overview of committing process.

In figure 6, `git commit` command committed a code to local repository with commit message “Hello”. Finally, such committed code needs to be uploaded to remote server, so that other coders can access the changes. For that purpose, `git push` command is used.

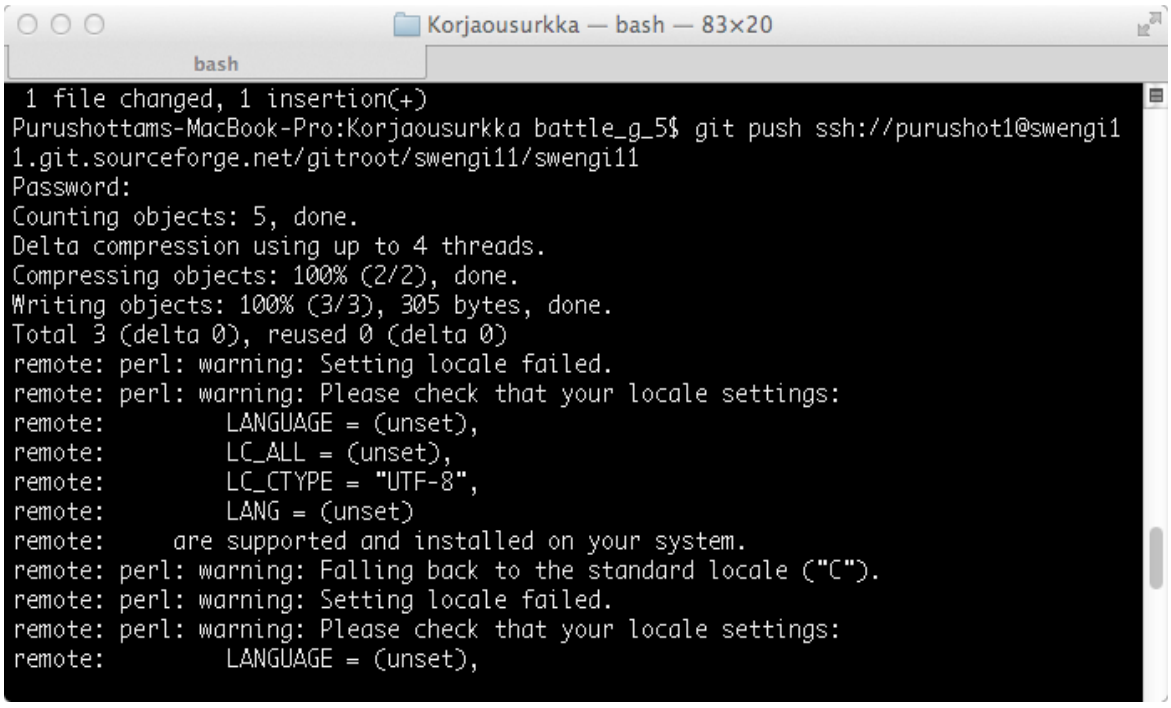

Figure 13. Demonstration of git push command.

In figure 13, `git push` command has been used to upload the local commit to remote repository. Hence, an application is backed-up in remote location and fellow developers can view and access it from any part of the world.

As discussed earlier, version control system saves project in such a way that changes are easily visible. VCS make use of different colors for that purpose. Git uses no different method than the color technique to make commits easily noticeable. The figure below depicts how git stores the changes in remote server.

Figure 14. Remote git repository.

In the figure above, texts that were added and removed from the project are in green and red color respectively. Furthermore, name of the committer and time of the commit are easily seen on the top.

Trac

A computer application is never complete or perfect. There is always something that can be added or made better. Computer software always demands change or optimization. Need of change may occur due to addition of new user group or expiration of technology that is being used. Change in computer software means addition of new functionality, replacement or fixing or enhancement of older functionality and deduction of existing feature. Reduction of existing feature may take place due to various reasons

such as unpopularity, to reduce operation costs or inappropriate content to users. In theory, modification in software is carried out to provide a better service. However, this is always not true and is kind of a gamble. Facebook introduced new timeline but many people totally hated it at first. Nonetheless, people liked the new photo gallery of Facebook. Facebook insisted on keeping timeline, thus it still exists and people are kind of got used to it. They do not make any complaints anymore. Presumed optimization of software can bring negative impact also, hence it needs to be carried out with great caution. Improper handling can delay or degrade the quality of software. In worst case, it may destroy the project completely. Thus, a tool is required for management and maintenances of change. Such tools or software is known as issue tracking system. It is also known as trouble tracking system, bug tracking system and requirements tracking system. However, the term issue tracking system is used in this paper to refer to any form of issue tracking system.

The main purpose of issue tracking system is listing use cases, managing them and monitoring their progress. It is a centralized system, where all members of development team can view the state of software. Information in issue tracking system is shared across each and every member of the team. Issue tracking system works in a fairly simple way. First of all, author creates an issue in a system. At this phase, an issue contains issue number, title, description, priority order, issue reporter's username and username of a person who it was assigned to. Issue number is just as same as serial number and increases by 1 whenever new issue is added. Title of an issue is a short form of description; it is written in a way that reader gets a basic idea of issue. Description on the other hand is a detailed explanation of an issue. Description has information like what needs to be created or fixed. Moreover, it describes the nature and behavior of bug. Furthermore, the right behavior of software is also highlighted. In description, problem is precisely mentioned. Likewise, all the necessary references are cited properly. Priority of an issue is set during creation process. Priority illustrates the level of urgency of an issue. The words chosen by different issue trackers to tell priority levels are different. However, It is always similar to words like urgent, major and minor. Issue tracker automatically saves the username of author. An author can assign an issue to specific developer, thus username of assigned developer is also visible. In some issue trackers, issues can be left as open and developers can choose from the list.

Once the issue has been assigned, developer works on it and continuously updates the status of that issue. For instance, the status of issue is always ‘To do’ when created. The status changes into ‘In Progress’ when developer starts to work on that issue. Once the issue is resolved, developers change the status to ‘Completed’. During this period, description of the issue is updated regularly. Updates contain information like solving strategy of issue, why one method over another was chosen to tackle problems and new findings on that matter. Issue tracker automatically saves the time when developer took issue and when it was marked as completed. The length of that period is also measured.

There are many issue-tracking systems available in the market. Some of the popular examples would be Bugzilla, Mantis, JIRA and Trac. Nevertheless, trac was used for this particular thesis project. Trac is a web-based issue tracking system that is implemented in python. It is an open source project, which can be used by anyone for free. It provides very easy and intuitive user interface. Furthermore, it is equipped with an interface to work with version control systems and integrated wiki. Due to these amazing qualities, it is very popular among open source communities. In trac world, a term ‘ticket’ is used to refer to issues. As an example, issue 154 is called as ticket 154. Trac is based on wiki, so user interface looks like a wiki page. Users quickly understand this

because welcome page is also a standard wiki page. All wiki tags work in trac environment. Users can use those tags during documentation also. The figure below exhibits the ticketing system of trac.

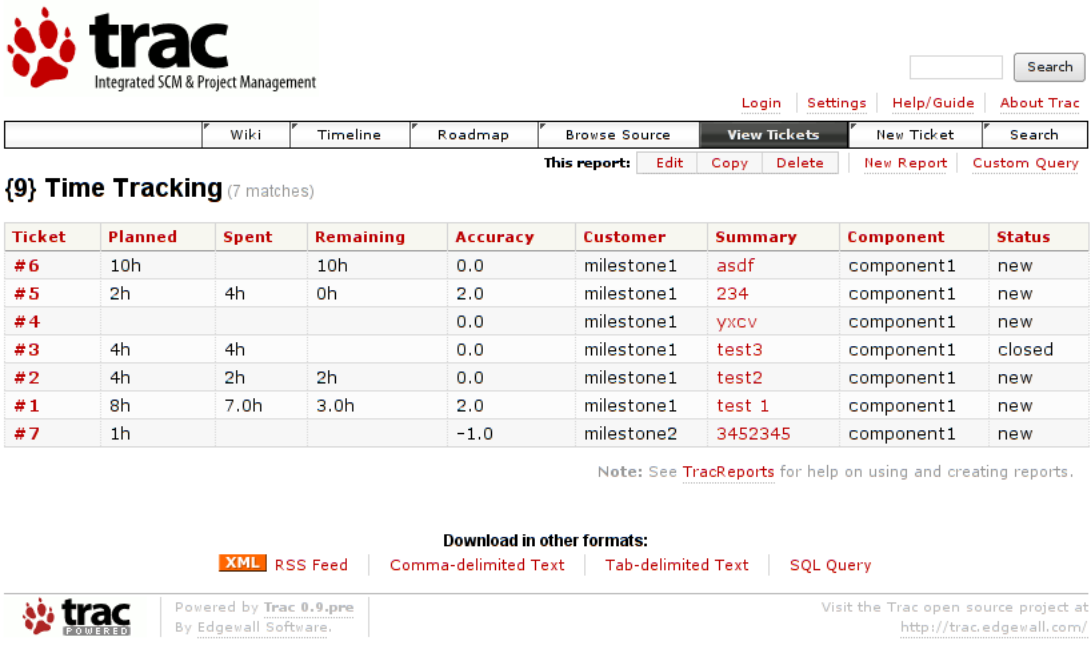

Figure 15. Overview of trac.[36.]

Figure 15 demonstrates the list of tickets in trac. Unique number is assigned to every ticket and they are listed in prioritized order. Trac is very convenient in tracking time because time planned, spent and remaining can be easily summed as shown in figure 15. In trac, milestones are almost like groups. There is list of issues in every milestone. Developers take issues in the ascending order from milestone. For instance, if there are three milestones like 1.1, 1.2 and 1.3; developers start to work with milestone 1.1. The usual practice is to put important items in earlier version of milestone. It is not necessary that every milestone have fixed number of items. Issues are always changing due to a shift in software requirements. It is also possible that milestone 1.3 have zero issues. In that case, one need to understand that planning for milestone 1.3 is not complete. Tickets will appear when milestone items are finalized. In general practice, fixed time is allocated for each milestone. The length of time varies from project to project; usually it is at least one month long.

Firebug

Firebug is a very popular free and open source web development tool developed by Firebug working group. It provides interface for debugging and editing a client side code live in any web service. It is an extension that can be integrated with Firefox. It does not support any other browser at the moment.

Monitoring of HTML, CSS, DOM and JavaScript code has become lot easier with the development of firebug. Furthermore, properties of HTML and CSS element can be modified to see the result instantly. Figure 16 illustrates the basic user interface of firebug.

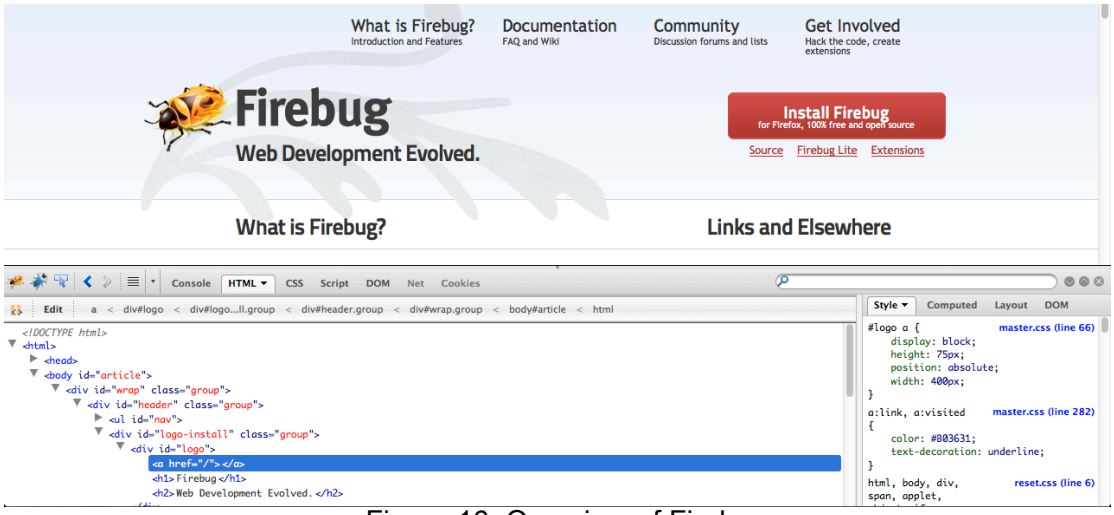

Figure 16. Overview of Firebug.

As we can see in figure 16, by using firebug we can easily check DOM structure of any webpage. Further, CSS styling used on page are visible in bottom right panel.

Skype

Although Skype is a communication tool used by general public, I would like to mention it here because communication is a big issue of distributed development. Skype and general emails were used as a communication means during the development of korjausurakka. Performance of Skype was really appreciable. It could easily handle the conference call between three people. On several occasions when product owner was also present in a meeting, the quality of the service did not decline. Hence, Skype is a very good tool for communication and can be applied professionally in distributed development. Further, Skype provides facility of video calling, which can be used to cre-

ate little more of a bonding than just an audio talk. However, we did not use video calling in this thesis project.

6.4 Parties involved

Korjausurakka is an interactive web-service. As a consequence, there are many parties involved in the service. As explained earlier, service seeker creates a job in the service. Once the job is online, contractors present their best offer to that job. Finally, service seekers choose the contractor of their preference. Now, it is very clear that there are two types of user groups involved. In addition to these two user groups, there is an administrator who monitors and manages the whole service. All of these roles are described below.

Service seekers

At first, Service seekers need to go into the service and create an account with valid information. They can do it free of charge. Once the user account is in place, they need to log in with valid credentials to be able to use the service. Service seekers have very limited privileges in the service. All they can do is create job and accept service providers. Service seekers need to provide full details of the job. Details contain time, price range and description of the job. Likewise, they can upload pictures, Microsoft Office and PDF files to make nature of the job clearer.

Job seekers see list of bidders to their job if any has been placed. Then, they can view the details of each and every offer. Finally, job seeker accepts the offer by clicking ‘Select offer’. Service seekers do not need to pay anything else than lump-sum money. Value added tax is also included in the offer.

After the completion of a job, service seekers can rate the performance of contractors. Ratings reflect the quality and attitude of a service provider. Better the rating better is the service provider. Ratings help service seeker to find a quality contractor.

Service Provider

Similarly, service providers also need to create a user account to be able to place bids in the service. However, they have to go through different path than service seekers. First of all, they need to pay to create an account. Second of all, an administrator has to process their request for creation of an account. They will only be able to log in when administrator verifies the integrity of information provided and member fee is received. Service providers are usually firms instead of an individual person. Administrator checks and verifies the registration number of a firm from a publically available database before granting an account. Once account is set, service providers can create their profile by adding description and photos.

All the available jobs are visible to service providers, hence they choose and place an offer to the relevant job. Once positive feedback is received from the job provider, contractors complete the job and receive a rating from service seekers.

There are three kinds of service providers depending on the membership fee. Ones paying less are basic members, middle ones are silver members and top ones are gold members. Basic members have very limited privileges. They can write limited characters for their description. Additionally, they can only upload several pictures. These privileges increase, as the membership status gets better. In addition to these privileges, gold members get benefit of online advertisement in the service.

Administrator

Finally, there is an administrator. Administrator is like a god within korjausurakka. He has all the privileges within the service. Primary job of an administrator is to manage user accounts and process requests coming from users. Requests usually come from service providers. Requests might be for creation or upgrade of an account. Furthermore, administrator can block or even delete the accounts if some malicious activities are found.

Project manager is the acting administrator of korjausurakka. He manages all the subscribers. Furthermore, he takes care of adding dynamic textual contents to a service.

7 Implementing Scrum in Distributed environment

As mentioned previously, Scrum was used as a software development method in development of korjausurakka. Nature of the project was distributed because all three developers were working from three different places. Senior developer was in Switzerland, whereas two junior developers were in Finland. Moreover, we were a totally integrated scrum team. Implementation of scrum in our project is described below.

7.1 Team involved in the project

Scrum team consisted four members. Every member had a specific role in a team.The team was more or less self-organized. The development team enjoyed flexibility of choosing tasks from product backlog. Moreover, product owner did not try to control the team. Personally, I never felt any kind of hesitation to interact or ask questions with either product owner or lead developer.

Product Owner

Product ownerof the project team was the sole stakeholder in the project. He provided all the resources and product backlog items. He briefed backlog items to development team, especially to junior developers because Dooxe framework was completely new to them.

Development Team

Development team consisted of three members: senior developer and two junior developers. The main tasks development team was assigned with were front and back end implementation on code level. There was not any graphical or structural designing to perform because we were only enhancing the already running application. Development team switched back and forth while developing front and back ends. Lead developer being the core coder of whole Dooxe framework had extensive knowledge about it. Hence, he helped juniors a lot to understand the framework. Moreover, he assisted in troubleshooting the technical problem.

Scrum Master

There was not any one assigned as a scrum master. However, lead coder led the daily scrum and performed some of the scrum master's responsibilities. On the other hand, product owner monitored the progress of entire project and fulfilled duties of project manager.

7.2 Events Occurred

The scrum events that occurred during the project are described below.

The Sprint

At the early stage of project, we had time-boxed sprint of one week. Nonetheless, due to poor understanding of Dooxe application on code level, we failed to deliver increment within allocated time. As a consequence, we modified scrum rules and deadlines were not made very strict. However, sprint of one to two weeks were employed.

Sprint Planning Meeting

Sprint planning meetings were held at Dooxe office. It was not a time-boxed event either, although, scrum rules say otherwise. It usually happened right after the completion of previous task. Sometimes, it occurred in middle of ongoing task to adjust the timetable of team members or to implement two tasks parallel manner.

Daily Scrum

Daily scrum meetings were held via skype. They occurred at eight o'clock in the morning. In daily meetings, senior coder gave solutions to our problem and suggested the most efficient approach to implement a task. Moreover, he showed us places where we should be looking at for coding backlog items.

Sprint Review

We had brief review sessions but it was more like item review rather than sprint review. Product owner did all the reviews and gave feedback on completed items.

7.3 Artifacts Produced

Scrum artifacts that are produced and relevant to this project are described below.

Product Backlog

Product backlog items were listed in issue tracking system. Detailed descriptions were provided for every item. They were continuously updated to meet the changing requirements.

Monitoring Project Progress

Project manager monitored the progress of a project. However, a common trend like burndown chart was not used. To monitor progress, project owner simply checked the number of remaining items in product backlog.

Sprint Backlog

Product owner selected sprint backlog items from product backlog. In addition, he assigned it to specific member of the development team. Likewise, he briefed and fine-grained those items before coding began.

Increment

The increment produced was relatively small. Sometimes it was as small as couple of lines of front-end codes. On contrary to the scrum principle, most of the increments produced at one sprint were not releasable. However, they were definitely usable. In addition, it should be kept in mind that sprint duration was much shorter than originally prescribed.

Definition of “Done”

Backlog items were marked as done when they satisfied all the test cases provided by product owner. Test cases were in description section of backlog. Moreover, they were briefed in sprint planning session. Product owner tested development items while development was in progress, which ensured their correct behavior. Product owner had the sole right of producing definition of “Done” and marking backlog items as completed.

8 Dooxe Retrospective

Dooxe was a very interesting project to be part of. First of all, the project was distributed then we were diving into an entirely new development environment. Nonetheless, improvements could have been made. There was plenty of room for enhancing productivity and efficiency.

Dooxe application did not have any documentation, which was a major obstacle in overcoming the learning curve. Lack of proper documentation significantly reduced the productivity at the initial stage of development. Thus, it is definitely a good idea to write and maintain documentation of an application. Wisely written document helps new developers to familiarize themselves with new system. I delivered the documentation for the part that I had developed in this project.

Likewise, in the early stage of development, implementation was carried out in a remote files, which slowed the development process. I personally do not enjoy editing remote files because changes take time to show results and there is no control over source code. If by any mistake typographical error occurs, then it is very hard to debug and it can corrupt the entire development environment. Moreover, it caused problems in switching between different versions of an application. I believe this project would have returned more increment if local environment was set up at initial stage. Nonetheless, it took more than a month to do it. Dooxe development environment in local machine was set up during lead developer's short visit to Finland. Local environment really helped in understanding the structure, objects, classes and methods of Dooxe framework.

Another problem that I realized was that the folder structure of Dooxe did not conform to the zend application, however, Dooxe was built using zend framework. In zend application, there should have been different folders for models, views and controllers. On the contrary, folder structure of Dooxe was completely different, which made the understanding of model, view and controller bit difficult. We were editing files at random while coding. Better understanding of these MVC entities would have yield fast and better increments.

Brief summary of findings in dooxe project are illustrated in table 3.

Findings	Comments
Distributed development	Communications helps to reduce the cultural difference. Continuous interaction between team members is important.
Small Company	Direct access to key person. Even junior developers can influence the decision making process. Today's software industry is heavily dependant on small companies.
SDLC models	They are not strictly followed in real life. These models should be modified to suit the need of project.
Documentation	SDLC models encourage to maintain proper document of software. Proper documentations help out the new developers.
Self-organizing team	Team spirit and commitment in self-organizing team is relatively high.
Small team	Communication is easy. Self-organization can be implied more effectively.
Scrum	Effective agile model for small teams. Productivity of scrum depends on the degree of self-organization in team. More the self-organizing behaviour better is the result.
Project monitoring	Project monitoring tools pushes the developers.

Table 3. Findings of dooxe project.

Table 3 is the concise representation of findings that were constructed in this dissertation. Despite these findings and limitations mentioned earlier, I think Dooxe project went pretty well. Although, I was an amateur developer with limited set of skills, I completed items that were assigned to me. It really was a memorable experience to be part of a real development team.

Likewise, on the face of these hindrances, project manager did a great job in keeping the team spirits up. I must say he did justice to the role of project manager and product owner.

9 Conclusion

Today's software industry will crumble without the contribution of small software houses. The development process is exactly similar except the number of manpower and resources available is limited in small companies. However, in small firms, there is huge room for developers to shine and show their ability because they have access to key people in a company. Thus, decision can be influenced and one's own idea can be planted.

As numerous computer systems are being developed, it is vital that safety and security needs are considered. Numerous catastrophic mishaps in the past have proved that software engineering principles need to be followed to ensure usability, efficiency, safety and security. As a consequence, considerable numbers of software development methodologies have been developed. These methods have their strengths and weaknesses. However, one must choose the method that suits best for their project. These methodologies prescribe strict guidelines. However, they are rarely followed in real life. In reality, they are always tweaked and bent to adapt to a needs of the project.

Among many software development methodologies, scrum is a free and versatile framework that can be employed not just to handle software projects but any project. It also provides the flexibility of using a certain portion of it. It is based on roles, events and artifacts, which are fairly simple to understand. It is one of the most popular methods in today's software business.

Furthermore, team development is a key to quality software. Team development creates a system of check and balance. Therefore, a mistake of one member can be detected and corrected by another member. Such approach will definitely produce a robust system. In addition, most of the software development teams are cross-functional in nature. Experts in different fields of software engineering have extensive knowledge of their related field and help yield more efficient results. Even if the team is not cross-functional, an extra hand is always a plus.

To meet the demands of growing software industry, a distributed approach of software development was introduced. Distributed approach helps in finding people with right skills. Furthermore, it is a great way of reducing software development costs. Outsourcing to a cheaper country is the ongoing trend in software industry.

To conclude, software development is a complex process. It always presents ambiguous future. Addition of distributed approach to that complexity makes it even more intricate. Such complexity can be overcome if strategy is planned carefully. Scrum is one of the most efficient battle-tested agile methods that can be applied to face the challenges posed by distributed development.

References

1. Bureau of Labor Statistics. Software Developers [online].
   URL: http://www.bls.gov/ooh/Computer-and-Information-Technology/Software-developers.htm#tab-1.
   Accessed 13 August 2013.

2. Aggarwal K. K., Singh Yogesh. Software Engineering. Delhi, India: New Age International Ltd; 2001.

3. Williams Laurie. A (Partial) Introduction to Software Engineering Practices and Metods. 5th Edition. 2008-2009.

4. Mohapatra P.K.J. Software Engineering. Delhi, IND: New Age International; 2010.

5. Burback Ron. Software Engineering Methodology: The Watersluice. 1998.

6. Lane M. Susan, editor. Object-Oriented Analysis and Design: Instructor Guide. Phoneix, AZ: ProsoftTraining; 2003.

7. Awad M.A. A Comparison between Agile and Traditional Software Development Methodologies. 2005.

8. Vliet Van Hans. Software Engineering: Principles and Practice. Wiley; 2007.

9. Lui Kim M. Chan, Keith C. Software Development Rhythms: Harmonizing Agile Practices of Synergy. NJ, USA: Wiley; 2008.

10. Munassar Ali Mohammed Nabil, Govardhan A. A Comparison Between Five Models of Software Engineering. International Journal of Computer Science Issues 2010;7(5):94-101.

11. Maza Resnick S, Bjork A.M. Professional Scrum with Team foundation Server 2010. NJ, USA: Wrox; 2011.

12. Goodpasture John C. Project Management the Agile way: Making It Work in the Enterprise. FL, USA: J. Ross Publishing Inc; 2010.

13. Martin Robert C., Martin Micah. Agile Principles, Patterns, and Practices in C#. NJ, USA: Pearson Education Inc; 2006.

14. Kelly Allan. Changing Software Development: Learning to Become Agile. Chichester, GBR: Wiley; 2008.

15. Beck Kent. Extreme Programming Explained.1st ed. 1999.

16. Holcombe Mike. Running an Agile Software Development Project. NJ, USA: Wiley; 2008.

17. GoyalSadhna. Agile Techniques for Project Management and Software Engineering.Technical University Munich; 2008.

18. Hunt John. Agile Software Construction. Wiltshire, UK: Experis Ltd; 2006.

19. AbeysingheSamisa. PHP Team Development: Easy and Effective Team Work Using MVC, Agile Development, Source Control, Testing, Bug Tracking, and More. Birmingham, GBR: Packt Publishing Ltd; 2009.

20. Cockburn Alistair. Agile Software Development: The cooperative game. Indiana, United States: Pearson Education; 2007.

21. Saksena A. Agile burn-down chart [online.]
URL: http://www.certschool.com/blog/agile-burn-down-charts/.
Accessed 7 July 2013.

22. Schwaber Ken, Sutherland Jeff. The Definitive Guide to Scrum: The Rules of the Game. 2011.

23. Highsmith Jim. Agile Software Development Ecosystems.Addison Wesley; 2002.

24. Mittal N. Self-Organizing Teams: What and How [online].
7 January 2013.
URL: http://www.scrumalliance.org/community/articles/2013/january/self-organizing-teams-what-and-how.
Accessed 9 September 2013.

25. Cohn M. The Role of Leaders on a Self-Organizing Team [online].
7 January 2010.
URL: http://www.mountaingoatsoftware.com/blog/the-role-of-leaders-on-a-self-organizing-team#comments.
Accessed 9 September 2013.

26. National Research Council Staff. Office Workstations in the Home. Washington DC, USA: National Academies Press; 1985.

27. Kile F. James. An Investigation into the effectiveness of agile software development with a highly distributed workforce. Pace University; 2007.

28. Alqahtani Abdullah Saad, Moore John David, Harrison David K, Wood Bruce M. The Challenges of Applying Distributed Agile Software Development: A Systematic Review. International Journal of Advances in Engineering and Technology 2013;5(2):23-36.

29. Jimenez Miguel, Piattini Mario, Vizcaino Aurora. Challenges and Improvements in Distributed Software Development: A Systematic Review. 2009.

30. Woodward Elizabeth, SurdekSteffan, Ganis Matthew. A Practical Guide to Distributed Scrum. Boston, MA: Pearson Education Inc; 2010.

31. DooxeOy. About Dooxe [online]. Helsinki, Finland: DooxeOy.
URL: http://www.dooxe.fi/info/about-dooxe
Accessed 3 August 2013.

32. Design Patterns: Model-View-Controller [online].
URL: http://cupsofcocoa.com/2011/08/13/
design-patterns-model-view-controller/
Accessed 14 August 2013.

33. Zend Technologies. About [online].
URL: http://framework.zend.com/about/
Accessed 14 August 2013.

34. Berkenalandengan Version Control [online].
URL: http://nopainsocounterpain.wordpress.com/2011/09/22/
berkenalan-dengan-version-control/
Accessed 14 August 2013.

35. Git [online].
URL: http://git-scm.com/
Accessed 14 August 2013.

36. Trac. Time Tracking [online].
URL: http://trac.edgewall.org/wiki/TimeTracking
Accessed 14 August 2013.