

Metamodelling and Evaluating Extreme Programming

Sundar Kunwar

University of Tampere
School of Information Sciences
Computer Science
M.Sc. thesis
Supervisor: Eleni Berki
May 2013

University of Tampere

School of Information Sciences

Computer Science

Sundar Kunwar: Metamodelling and Evaluating Extreme Programming

M.Sc. thesis, 70 pages

May 2013

Agile software development methods have drawn the attention of software development professionals in the past few years. Agile software development methods use iterative and incremental approaches to address the changing requirements of customers. One of the well-known agile software development methods is extreme Programming (XP) and is derived by sets of values including simplicity, communication, feedback and courage. The extreme practices, variation in composition and interaction between values and the feedback in XP have made the software system more complex and demands the improvements and evaluation framework to understand and evaluate the XP practices in a practical way.

The main aims of this study are to improve some of the extreme practices of XP through agile modeling and evaluate the XP projects using XP evaluation framework. Two research questions were set to find out the enabling and limiting factors of extreme practices of XP and the way to improve the XP software process. An interpretive research approach was used to conduct a literature review to develop the agile meta-models and evaluation framework for process improvement. The contribution of thesis work can be broadly categorized into two parts. The first part deals with modelling the three most criticized and extreme practices (lightweight requirement, Pair Programming and onsite customer) of XP and the second part is concerned with developing the evaluation framework for XP. Use cases are collected from scenario based requirement engineering practice with stakeholder analysis to address the lightweight requirement of XP. Problems of Pair Programming are addressed by personal development traits, Distributed Pair Programming (DPP) and Collaborative Adversarial Pair (CAP) Programming models. Surrogate customers and multiple customer models are two alternatives proposed to address the problems of onsite customer in XP. The XP evaluation framework is a collection of some new and validated metrics used for evaluating XP projects, XP practices, XP products and some additional factors concerned with XP.

Key words and terms: Agile, extreme Programming (XP), interpretive research, Collaborative Adversarial Pair (CAP) and extreme practices.

Acknowledgement

My thesis work would not have been possible without the proper guidance and the sincere help from several persons who in one way or another contributed and provided valuable suggestions and feedback in preparing and completing the thesis work.

Foremost, I would like to express my sincere gratitude to my professor and thesis supervisor Eleni Berki for the valuable guidance and continuous support to my thesis work. I am thankful for her patience, immense knowledge and motivation. Her guidance helped me a lot all the time in writing my thesis.

Besides my supervisor, I would like to thank to my professor Zheyang Zhang for arranging the thesis seminar and for providing creative comments on the idea and progress presentation. I would also like to thank all the attendee of the thesis seminar for paying attention to my thesis work and for providing the creative and reflective comments. I would also like to thank my English teacher Robert Hollingsworth who helped me in making familiarization with thesis writing and academic writing. I would also like to express my sincere thanks to project teacher Timo Poranen for his constructive feedback.

Last but not the least, I would like to thank all my family members, friends and relatives who encouraged and helped me in all the way for making my study possible.

Sundar Kunwar

15th May 2013, Tampere

Table of Contents

1. Introduction	1
1.1 Extreme Programming (XP).....	3
1.2 Scrum.....	4
1.3 Related Work.....	6
1.4 Research Method	7
1.5 Thesis Contribution	8
1.6 Thesis Structure	8
2. Rules and Practices of XP	10
2.1 Whole Team	10
2.2 Planning Game	10
2.2.1. Release Planning	11
2.2.2. Iteration Planning	11
2.3 Customer Tests	12
2.4 Releases	12
2.5 Simple Design	13
2.6 Pair Programming.....	13
2.7 Test-Driven Development	13
2.8 Design Improvement	14
2.9 Continuous Integration	14
2.10 Collective Code Ownership.....	15
2.11 Coding Standard	15
2.12 Metaphor	15
2.13 Sustainable Pace	16
3. Modelling Approaches	17
3.1 System Dynamics	17
3.2 Computer Simulation	17
3.3. Agile Modelling (AM)	18
4. Pitfalls of XP	21
4.1 Requirement	21
4.2 Onsite Customer	23
4.3 Pair Programming.....	23
5. Addressing Pitfalls through Agile Modelling.....	25
5.1 Requirement Model	25
5.2 Onsite Customer Model.....	33
5.3 Pair Programming Model	35
6. Evaluation of XP	40

6.1 Meaning of Measurement.....	40
6.2 Software Metrics	41
6.3 Proposed Evaluation Framework for XP.....	42
6.3.1. Project Records	43
6.3.2. XP Practices Metrics	46
6.3.3. XP Product Metrics	50
6.3.4. XP Additional Metrics	52
7. Discussion.....	54
8. Conclusion.....	63
References	65

1. Introduction

Software development approaches have been enhancing significantly all the time. It simply means that software development methodologies are expanding and are becoming more complex because software engineering is merged with different diverse fields. Software development methodologies are the frameworks that are used for structuring, planning and controlling the processes involved in software development. Traditional software development methodologies are plan driven heavyweight methodologies because they consist of sequential series of steps that need to be planned and documented in detail before implementation. The waterfall model, V shaped model and Rational Unified Process (RUP) are the most popular traditional software development methodologies. A lot of money is spent on developing these methods that have nothing to do with the customers' requirements. This will certainly increase the cost of the product. Most of the traditional software development methodologies are very rigid to change. Changes are only possible if the improvements are brought back to an earlier stage which is waste of time, money and resources. In simple words, they are not able to address the changing requirements of the market any more. As a result, new software development approaches have evolved as agile methodologies to address the rapid changing requirement of the market. According to the Merriam-Webster [2012] online dictionary, 'agile' is defined as the ability to move quickly and having the characteristics of being easy, adaptable and resourceful. In agile software development, 'agile' means the ability to respond to change. Therefore, it is not simply the size of the process or the speed of delivery; it is about the flexibility of the process or methods [Kruchten, 2010]. Kruchten advocates agility as flexibility and adaptability, but according to Cockburn [2001], "Core to agile software development is the use of light but sufficient rules of project behaviour and the use of human and communication oriented rules."

Agile methodologies are the reactions to the traditional methods with documentation driven and heavyweight software development processes. Agile methodologies include modification in software development process to make them faster, more flexible, lightweight and productive. In the late 1990's, several software development methodologies drew the attention of the public and each method has a combination of old ideas, new ideas and transmuted old ideas [Kalermo & Rissanen, 2002]. What was common among all these methodologies was that they all emphasized personal interaction over process, direct communication, short and frequent release, iterative process, self organization and code crafting among others.

At a summit of seventeen independent practitioners of several programming methodologies, the agile manifesto was written in February of 2001. The practitioners were found consensus around the following four values:

- i. Individuals and interactions over processes and tools
- ii. Working software over comprehensive documentation
- iii. Customer collaboration over contract negotiation
- iv. Responding to change over following a plan [Agile Manifesto, 2001]

In addition to the four values, the Agile Manifesto also contained the following twelve principles:

- i. Customer satisfaction and continuous software delivery are given high priority.
- ii. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- iii. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- iv. Business people and developers must work together daily throughout the project.
- v. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- vi. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- vii. Working software is the primary measure of progress.
- viii. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- ix. Continuous attention to technical excellence and good design enhances agility.
- x. Simplicity-the art of maximizing the amount of work not done--is essential.
- xi. The best architectures, requirements, and designs emerge from self-organizing teams.
- xii. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly. [Agile Manifesto, 2001]

There are many agile methodologies in use today. Some of the most popular ones are extreme Programming (XP), Scrum, Feature Driven Development (FDD), Crystal Methodologies Family (CMF) and Adaptive Software Development (ASD). FDD is an iterative and incremental agile software development method. Its practices are all driven by a client-valued functionality known to be feature. CMF is the collection of

lightweight methodologies. ASD is a software development process characterized by rapid software development. The continuous adaptation of the process to the work at hand is the normal state of affairs in ASD. Of these, XP and Scrum are the most commonly used agile software development methodologies.

1.1 Extreme Programming (XP)

One of the well-known methods of Agile is extreme programming (XP in short) and is driven by a set of values including simplicity, communication, feedback and courage [Beck, 1999a]. The XP process is characterized by a short development life cycle, incremental planning, continuous feedback and reliance on communication and evolutionary design. The core part of XP consists of a simple set of practices including a planning game, small releases, metaphor, simple design, test driven development (TDD), refactoring, Pair Programming (PP), collective ownership, continuous integration, 40 hour week, onsite customer and coding standards [Sfetsos et al. , 2006]. This interesting composition of XP is one of the main reasons that make it successful.

It is necessary to identify and handle the problems such as complexity, conformity, changeability and invisibility in each type of software process improvement. According to the software engineering theory, the first question that the company should ask is what the problems with our current process are and then go for improvements and changes if necessary [Louridas et al., 2008]. The processes of the system are mainly dynamic in nature with the involvement of humans as managers, developers and customers among others. Variation in human observation variation, instruction, communication, interest, culture, experiences, and inclination make the process more complex and dynamic [Yong & Zhou, 2009]. It was first Beck [2000] and Jeffries et al. [2001] who developed the XP as system. Extreme Programming (XP) is a software development methodology developed to improve the quality of software as well to respond to the changing needs of customers. Broadly, it advocates short and frequent small releases to improve productivity and introduces lightweight practices in software development methodology. It is known from different studies that the ability to successfully implement the XP process varies from company to company and is heavily based on tacit knowledge, skill, frequent communication and motivation. Beck (1999a) stated that XP is an intensely social activity and not everyone can learn it. However, the variation in composition and interaction between the values and practices and their feedback in the XP system has made the software system more complex and needs more knowledge to understand each and every common practice of XP [Beck, 2000]. XP is known to be a lightweight agile software development methodology with some extreme practices which are lightweight in nature but very difficult and

sometimes unrealistic to implement practices and there are only a few analytic studies related to XP. Most of the literature and books have been drafted by the inventors of the Agile Manifesto and are concerned with the promotion and commercialization of the agile methods and the services they provide. Therefore, most of the materials seem like promotional material rather than an analysis of strength and weakness as of agile software development. Figure 1 shows the general overviews of XP. Release planning is done with the help of system metaphor obtained architectural spikes and the requirement specifications obtained from user story. Release plan helps to carry out the iteration which in turn produces a piece of software. Small releases are released after the acceptance test approved by customer.

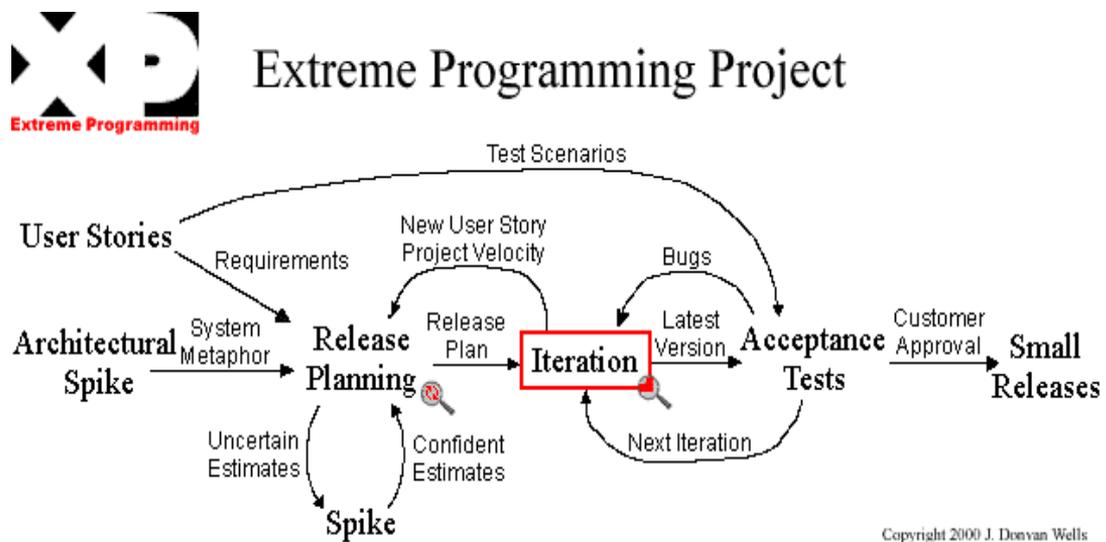


Figure 1: XP release cycle [XP flow Chart, 2013].

1.2 Scrum

Jeff Sutherland created the Scrum process in 1993 [Sutherland, 2004]. The name Scrum was borrowed from an analogy put forward by a study carried out by Takeuchi and Nonaka. They have compared high-performing, cross-functional teams to the Scrum formation used by rugby teams in their study. [Takeuchi & Nonaka, 1986] Scrum is a lightweight process used for managing and controlling software and the software development process. It was found that Scrum was practiced before the announcement of the Agile Manifesto. It was later included into agile methodology because of the same underlying concepts and principles. Scrum shares the basic concept of agile methodologies with some project management practices. It is a leading agile software

development methodology used by many fortune companies around the world and the Scrum framework consists of the following components [Scrum Alliance, 2013]:

-A product owner creates prioritized wish lists called ‘product backlogs’. They are a simplified form of requirement list.

-Some wish lists, most possibly the higher prioritized wish lists, are selected to be implemented in each sprint, and sprint planning is carried out to decide how to implement those wish lists.

-A team has a very short time to implement those wish lists and the duration is called ‘sprint’ and generally sprint duration lasts for two to four weeks. Daily meetings are carried on to know the problems and progress.

-A team leader is called a ‘Scrum master’, who is supposed to focus all the team members on the Scrum goal.

-At the end of the sprint, it is supposed that the implementation of the wish lists should be ready to show to the client.

-Finally, the sprint ends with a sprint review and retrospective.

-The next sprint is carried out with the same rules but with different wish lists to implement.

As explained above, the general overview of the Scrum development cycle is shown in Figure 2. The development cycle repeats until the logs in the product backlog have been successfully completed, the budget depleted and the deadline arrives. Scrum makes sure that the most of the prioritized tasks have been completed before the termination of the project.

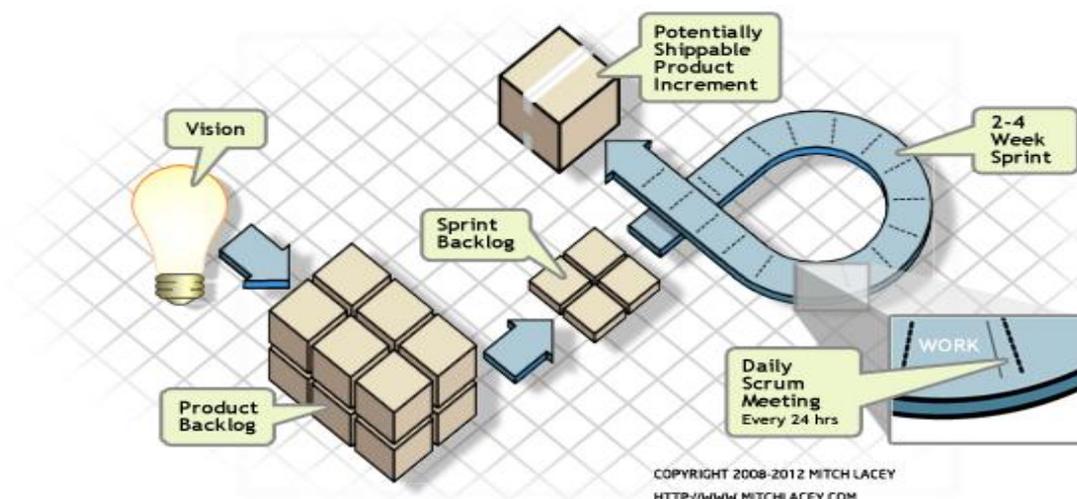


Figure 2: Scrum Overview [Scrum Alliance, 2013].

1.3 Related Work

Extreme programming (XP) is widely used in both academic and industry sectors, but there is only little work done on XP modelling and evaluating to improve the XP software process. It is very difficult to implement all the practices of XP. A study was carried out to know about the existing models of XP and what kinds of models are necessary for future work [Abouelela & Benedicenti, 2010]. System Dynamic model of XP development process was used to evaluate the development process quantitatively and XP practices by simulation [Yong & Zhou, 2009]. The controlled experiment with students was carried out to find the effect of Pair Programming. The students were divided into groups to find the effects of Pair Programming in XP. Four experiments were carried out to find the effects of Pair Programming at Poznan University of Technology. [Nawrocki & Wojciechowski, 2001] After the development of integrative models of software development project management, Wernick and Hall studied the impact of Pair Programming on the long term evolution of software systems [Wernick & Hall, 2004]

A quantitative evaluation framework was proposed for agile methodologies. The proposed evaluation framework measures the agile methodologies based on the postulates from Agile Manifesto. For each methodology, four postulates and corresponding formula are used for quantitative evaluation. [Karla et al., 2010] The methodologies can be evaluated and constructed using evaluation frameworks and meta-models and they are referred as meta-methodologies. A comprehensive overview of building of efficient and cost effective meta-models and evaluation frameworks with qualities properties identified in scientific and reliable way was provided. [Berki, 2006] Williams initiated the evaluation framework for XP as a part of her Software Engineering Research group work in empirical software engineering and it consists of three parts-context factors, adherence measures and outcome measures [Williams et al., 2005].

There is no evidence of modelling the XP (building the model of XP) to address its major pitfalls. Most of the works were only focused on finding the pitfalls of XP and provided some alternative solutions by comparing with other methodologies such as XP vs. Capability Maturity Model (CMM), XP vs. Sommerville-Sawyer model and XP vs. Scrum among others [Nawrocki et al., 2002]. The quantitative framework works only on four postulates based on the Agile Manifesto and provides the quantitative values based on that postulates. Evaluation Framework developed by Williams [2005] is more general and is not only focused to XP. It is a more generalized form of an evaluation framework for agile methodologies. Therefore, there is a need for more XP focused

evaluation framework that basically concentrates on XP practices, XP product and XP project.

1.4 Research Method

My aim is to gain a thorough understanding of current practices of extreme programming and to build the agile models of extreme practices addressing the pitfalls of XP and I also propose the evaluation framework that best suits XP practices. My work is more concerned with the applicability of XP. I have considered Extreme Programming as an initial research framework for explaining and evaluating various aspects of it.

The research questions derived from the research intentions are:

1. What are the enabling and limiting factors of extreme practices of XP?
2. How could it be possible to improve the XP software process?

To attempt to answer the research questions set, I will follow an interpretive approach to conduct a literature review. A research can be interpretive if it builds on the assumptions that humans learn about the reality from the meanings they assign to social phenomena such as language, consciousness, shared experiences, publications, tools, and other artefacts [David, 2010]. The most fundamental principle of the interactive research approach is a hermeneutic cycle derived from documents and literary analysis. The different components of the hermeneutic cycle are illustrated in Figure 3.

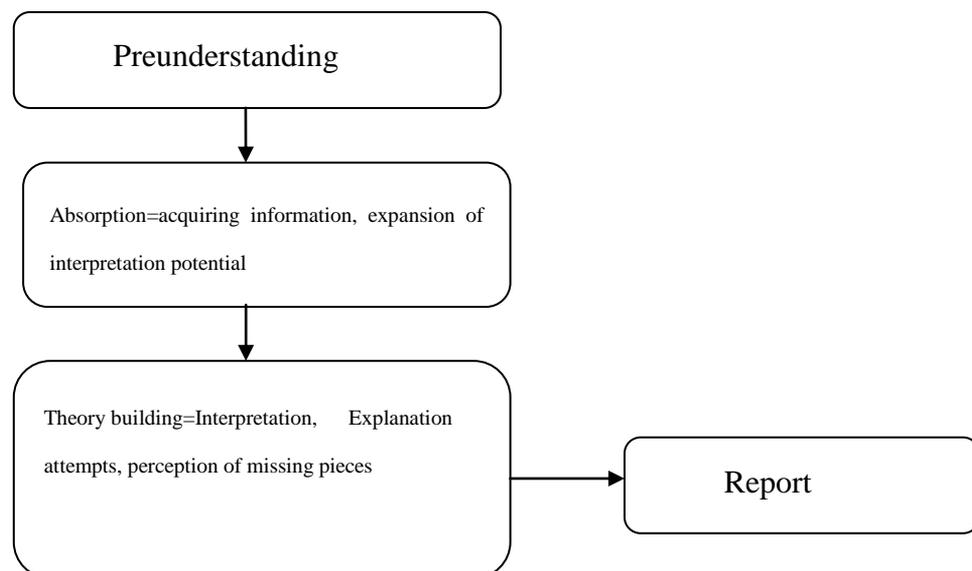


Figure 3: Hermeneutic cycle adapted from [Tamminen 1992, 95].

The first component of the hermeneutic cycle is concerned about the pre-understanding of researchers on the subject matter and the second component is concerned with the absorption of more knowledge from different sources to widen knowledge to expand the researcher's interpretation potential. The third component is concerned with theory building on the basis of an interpretation of knowledge, explanation attempts and missing knowledge. The last component is concerned with documenting the new theories and knowledge acquired through interpretive research approach. [Kalermo & Rissanen, 2000] The same approach of the hermeneutic cycle will be used for modelling and evaluating Extreme Programming.

1.5 Thesis Contribution

I have used Extreme Programming as my research framework to examine the causes why 100 percent implementation of XP is not possible and how XP can be evaluated in an effective and efficient way. Therefore, I followed an interpretive approach to conduct the literature review and this approach is concerned with the hermeneutic cycle derived from document and literary analysis. I used the hermeneutic cycle for modelling and evaluating extreme programming regarding the most criticised practices of XP. My contribution can be broadly categorized into two sections:

- i. Modelling the most criticised and extreme practices of XP.
- ii. Developing XP focussed evaluation framework.

Lightweight requirement, onsite customer and Pair Programming are the three most criticised and extreme practices of XP. Interpretive approach helped me in agile modelling to address all the pitfalls of the three extreme practices of XP to make it realistic and practical. The same approach was used in developing the evaluating framework that is concerned with XP. Speaking more precisely, my contributions are as follows:

- i. Investigate the most criticised and extreme practices of XP.
- ii. Make XP practitioners more careful in adopting all the extreme practices of XP.
- iii. Find out the solutions for the most criticised and extreme practices of XP.
- iv. Avoid risk for adopting XP practices.
- v. Provide a basic idea for adapting the improved practices using agile modelling.
- vi. Develop evaluation framework helps to evaluate XP project.

1.6 Thesis Structure

My thesis is structured as: Chapter 1 includes an introduction of traditional and agile software development methodologies. It also includes the related work, research

method, contributions and the structure of the thesis. Chapter 2 explains the rules and practices of XP. It also shows how these practices are interrelated to each other. Chapter 3 includes the possible modelling approaches that can be employed with XP practices. It explains dynamic modelling, computer simulation and agile modelling approaches. It clarifies why an agile modelling approach is suitable to XP. Chapter 4 includes the explanation of three most criticized and extreme practices of XP-lightweight requirement (user story), onsite customer and Pair Programming. Chapter 5 includes the solutions to those criticisms in order to eliminate or reduce them. A scenario based requirement is presented as an alternative solution to XP lightweight requirement, multiple customers and surrogate customer is presented as alternative solution to XP onsite customer practice; and distributed Pair Programming and collaborative Pair Programming is presented as an alternative solution to XP Pair Programming practice. Chapter 6 explains the need for and development of the XP evaluation framework. It includes various validated and few new metrics for XP evaluation. Chapter 7 includes the discussion about the work done. It also analyzes the result. Chapter 8 concludes the thesis work. It also includes the limitations of the study and the work that can be extended in the future.

2. Rules and Practices of XP

XP is the lightweight methodology for software development and is oriented towards the delivering the incrementally growing software products [Yong & Zhou, 2009]. XP is flexible in nature because strong preference is given to the informal communication of the development team over written documentation. The rules and practices used in XP are described below. The interrelations among XP practices are shown in Figure 4:

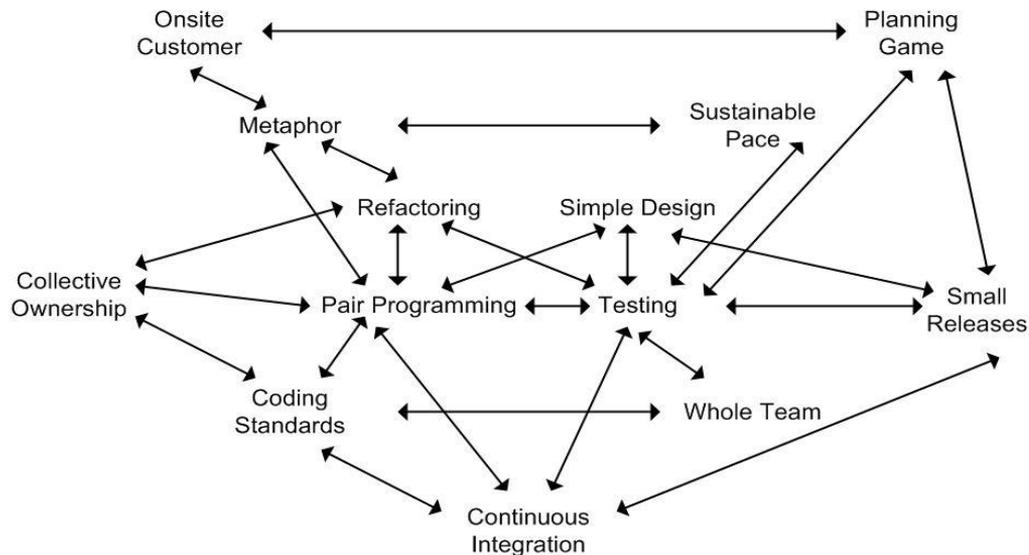


Figure 4: Interrelation among XP practices [Beck, 1999b].

2.1 Whole Team

The whole team includes all the contributors to an XP project who sit together as members of one team. This team includes the customer representative who is responsible for providing the requirements, priorities and the feedbacks, programmers who are responsible team members in implementing the customer's requirements, testers who are responsible for helping the customer to define the customer acceptance tests, coach or project manager for team management, resource allocation, handling the external communication, coordinating activities and facilitating the process for smooth operation [Kalermo & Rissanen, 2002]. The best team has no specialists but only contributors.

2.2 Planning Game

The main purpose of the planning is to determine what will be done by the date and then what will be done after that. It consists of three phases-Exploration phase, Commitment phase and Steering phase. Customer and development team go through first two phases and after the team has committed to release plan, steering phase

commences. The main objective of the planning is steering the project and giving the right direction towards its goal. The general block diagram illustrating the three phases of planning game is shown below:

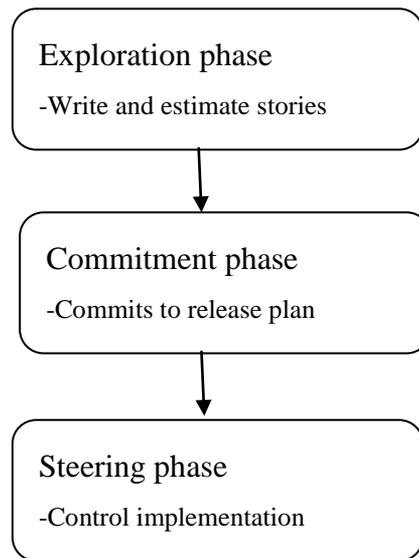


Figure 5: Three phases of Planning Game.

These three phases can be carried are two planning steps in XP and they are [Kalermo and Rissanen, 2002]:

2.2.1. Release Planning

In release planning the customer presents the desired features to the programmers and the programmers estimate the difficulty of the release plan. The customer lay out the plan without the cost estimates and knowledge of the important features. Initial release plans are generally imprecise and XP teams revise the release plan regularly to make it more precise and accurate.

2.2.2. Iteration Planning

Iteration planning is mainly concerned with giving right directions to team members frequently. So, XP team releases small release in every two week iterations and working piece of software is delivered at the end of the iteration. The customer puts forward the desired features to be implemented by next iteration. It is the tasks of programmers to break them into manageable tasks and estimate their costs. Team members decide the tasks to be performed in current iterations based on the tasks accomplished in the previous iteration. The number of days and the user stories completed in an iteration is expressed in term of project velocity. Simply, it measures

the length and the tasks completed in an iteration. The diagram shown in Figure 6 illustrates how iteration planning is carried out in XP.

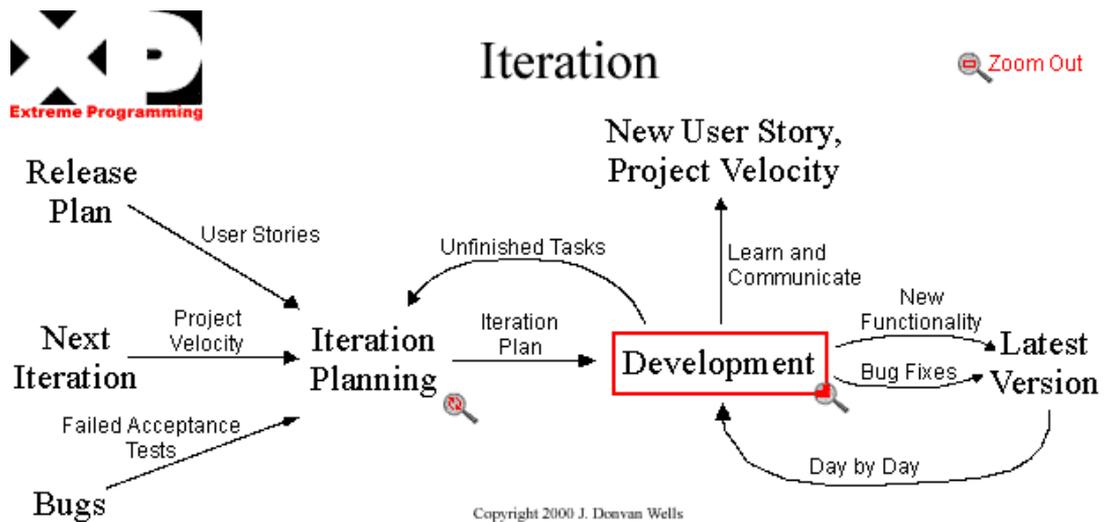


Figure 6: Iteration Planning [XP flow Chart, 2013].

The planning steps are very easy and simple with valuable information and the best part of this type of planning is that the control is in the hand of the customer. The project progress is clearly visible after two weeks of time. There is no provision to see the percent of work done. It tells about the completeness of user story. The customer has a right to cancel the progress of job if a customer thinks that it is not sufficient. Therefore, customer satisfaction is addressed properly in XP. XP projects are more concerned with the delivering the more with less stress and pressure.

2.3 Customer Tests

The XP Customer is responsible for defining one or more automated acceptance tests to ensure that the desired features are working properly. These tests are built by the XP team and use them to make sure that the implemented features are built according to wish on the customer. The customer has full authority to accept or reject the implemented piece of software. An automated acceptance test plays an important role to skip the manual test which saves time, money and effort. Automated acceptance tests are always treated like a programmer tests.

2.4 Releases

The XP team makes a practice of small releases in two ways: In all iterations, the team release running and tested software to business value recommended by the customer

and the customer can use this software for evaluation or can even release to end users. The team work is visible and the customer is responsible for evaluation. The XP team also releases software to their end users frequently. In XP Web projects releases are often done daily, in house project monthly and more frequently.

2.5 Simple Design

XP teams make the software simple with adequate design. They start with simple and through programmer testing and design improvement, it is iterated many times to refine the design. The XP teams make sure that the current design suits the current functionality of the system. The design in XP is not a onetime process but it is all the time process. The design is focussed throughout the whole process of development.

2.6 Pair Programming

Pair Programming (PP) in XP is a software development practice with two programmers working at single work station and one is a driver who writes the code while another is the observer who reviews each line of codes and their roles switches frequently [Williams et al., 2000]. It may be thought that it is inefficient to engage two programmers for the same job but at the same time the reverse is true. Some studies have shown that PP is more effective than traditional programming while other studies have shown that PP is not always practical due lack of resources like small team and also due to lack of developer's interest.

2.7 Test-Driven Development

Extreme Programming is facilitated with feedback loops. In the software development process, good feedback requires good testing procedures. In XP, a test is added to each short cycle before the coding has started to make it work with the code. This practice is known as Test Driven Development (TDD) to make sure all the codes are covered with tests. The diagram below shows activities concerned with the test driven development [XP flow Chart, 2013].

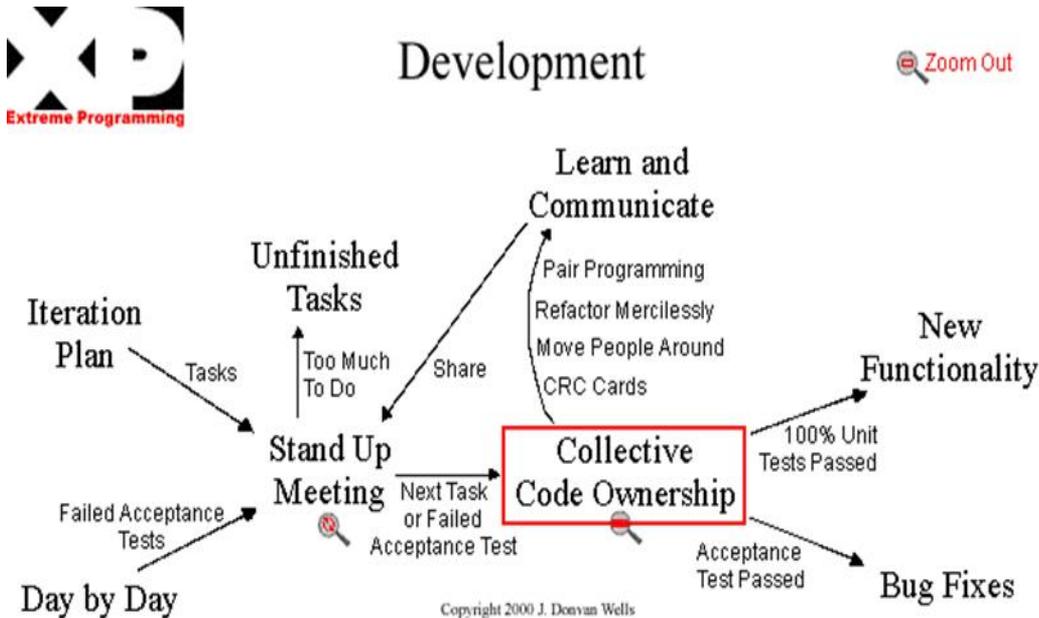


Figure 7: Test Driven Development [XP flow Chart, 2013].

2.8 Design Improvement

XP team puts continuous effort on delivering the business value in all iterations. The software should be well designed to deliver business value to the customers. Therefore, XP uses continuous design improvement process called refactoring as explained in a book called Refactoring: Improving the Design of Existing Code. [Fowler et al., 2002]

Refactoring is the process that focuses on the removal of duplication which is a sign of poor design, and helps to increase the cohesion of the code lowering the coupling at the same time. High cohesion and low coupling are recognized as the hallmarks of well-designed code for at least thirty years. XP always starts with the good and simple design. Refactoring is strongly supported by the comprehensive testing to be sure that the design is well prepared. Thus, the customer tests and programmer tests are the critical enabling factor. [XP flow Chart, 2013]

2.9 Continuous Integration

The system developed using XP is fully integrated all the time. There are multiple builds in XP projects. If there is no continuous integration in XP, there arise serious problems in a software project. Continuous integration plays important roles in delivering good quality work to the customer. The problems that appear after integration are avoided by practicing continuous integration of work. [Fowler, 2006]

2.10 Collective Code Ownership

Collective Code Ownership is one of the widely accepted practices of XP where everyone can contribute new ideas to the project. Any developer is free to edit, add, fix bugs and improve designs in the project. No one acts like a bottle neck for making changes. Design for next task or failed acceptance test are done with the help of Component Responsibility Collaborator (CRC) card. CRC card is used in object oriented software for brainstorming the object-oriented design. The diagram below shows how the code has relation with other different activities in XP. [XP flow Chart, 2013]

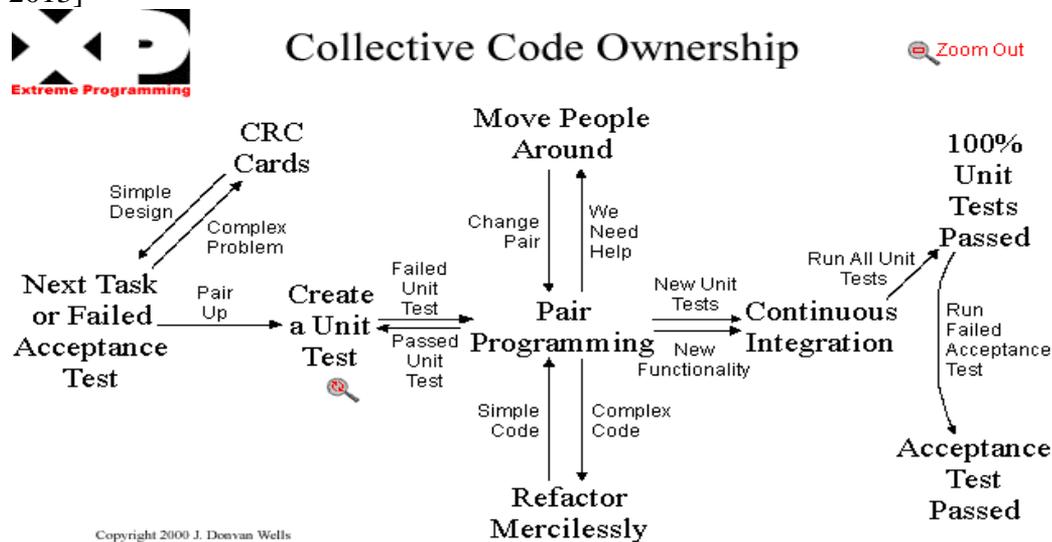


Figure 8: Pair Programming relationship [XP flow Chart, 2013].

2.11 Coding Standard

XP team follows common agreed coding standards which help to keep code consistent and easy to understand, read and refactor. The codes look like neatly written by a single competent developer that helps to encourage collective code ownership. [XP flow Chart, 2013]

2.12 Metaphor

The common vision which determines how the program should work is called a metaphor. It is more concerned with simple design with certain qualities. It helps to make the methodology lightweight. It is also concerned with a consistent naming method for classes and functions. [XP flow Chart, 2013]

2.13 Sustainable Pace

Sustainable Pace in XP helps to plan the releases and iteration. It also helps to determine the perfect project velocity that will remain consistent for the whole project. The pace is determined in such a way to maximize the productivity. XP team is for winning, not for dying. [XP flow Chart, 2013]

3. Modelling Approaches

The common purpose of modelling is to provide a basis for deeper understanding with experiments, predicting the behaviour of the system and saving the cost of actual case controlled experiments. There are various techniques and strategies to model the behaviour of the system.

3.1 System Dynamics

The methodology to analysis the situation that changes over time is system dynamics. Forrester developed system dynamic in 1951 at MIT. It was used for analyzing the interrelationship of the world's economy and the environment. It was promoted by its own society, conferences and publications [Hayward, 2000].

System dynamic approach is used in complex systems which are dependent, contains feedback loop, interaction and circular causality. It has already shown good analysis in applied economics, environmental science, industrial management, theory building process and many other fields. There are two tools which are widely used in system dynamics. Stock and flow diagram is used for system structure representation and causal loop diagram is used for visual representation of feedback loop [Yong & Zhou, 2009].

In this approach system are defined dynamically by graphical representation over time. Basically, the systems are represented by first order differential equations. To represent the system mathematically, one need to have a deeper understanding about the dynamics of the system and should have a deeper knowledge of mathematics.

3.2 Computer Simulation

Computer simulation is the process of designing the model of the real system and then implementing the model with a computer program for the purpose of conducting experiments to understand the behaviour of the system or to evaluate the operations or processes of the system. Computer simulations are the means to get answers about what if question from different stakeholders of the system. A system can be classified as stochastic or deterministic based upon the degree of randomness behaviour of the system. A system is said to be stochastic if the system is concerned with random behaviours and conversely, deterministic system is not based on the random behaviour. Based on the activities occurring in the system, it can be classified as a continuous or discrete system. [Melis, 2006]

i. Continuous System

In continuous system, system behaviour is modelled as the sequence of events that changes continuously with time. For example, the change in supply chain of product through time. Smooth changes in continuous time are focused rather than individual events. It is modelled using the smooth changes of the variable with the help of suitable continuous equation and then implemented using computer program. This type of simulation is known as a continuous system simulation. [Pidd, 1994]

ii. Discrete System

In discrete system, entity's behaviour is modelled as the sequence of events which state changes with point of time. For example, a customer in the bank may arrive (event), he/she get services from the bank (event), service will end (event) and so on. Modelling is done to capture the behaviour by distinct logic of these events and implemented using computer program. This kind of simulation is known as discrete system simulation. The time interval for discrete event is irregular and is modelled using the concept of random number generation. The irregularity of the time interval of events leads to the stochastic behaviour of the system. [Pidd, 1994]

3.3. Agile Modelling (AM)

An initial group of seventeen different methodologies was formed to address the challenges of software development and changing requirements of customers and is called Agile Software Development Alliance (www.agilealliance.org) and later it was simply referred as Agile Alliance [Ambler, 2002]. The interesting fact was that all the group members came from different background and agreed on the issues that the methodologies did not agree on [Fowler, 2000]. This group defined the manifesto to encourage the better way of developing the software and based on the manifesto, the criteria for agile software development such as Agile Modelling was introduced for the first time [Ambler, 2002].

Agile Modelling (AM) is the chaotic, practice based methodology for effective modelling and documentation of software based systems. AM methodology is the collections of practices guided by the principles and values for software professionals for applying on day to day basis [Ambler, 2002]. AM does not tell about how to build the model, but it tells about how to be effective as modelers. In other word it is not prescriptive process. AM is chaotic because it blends the chaos of simple modelling practices and blends it with the order inherent in software modelling artefacts. AM is simple, fast and touch freely modelling approach and anyone can do it. It is more art than science.

Agile software development methodologies such as eXtreme Programming (XP), Scrum and Dynamic System Development Method (DSDM) effectively use the modelling activities. Some of the most common modelling techniques in XP are user stories, Component Responsibility Collaborator (CRC) cards, and sketches for other different activities. Prescriptive software process such as Unified Process (UP) also effectively uses the modelling activities. There are Agile Modelling has specially three goals [Ambler, 2002]:

- i. It is used for defining and showing how to put into practice the collection of values and principles in lightweight modelling. Modelling techniques such as a use case model, data model and interface model acts as a catalyst for clear understanding of a system and its improvement.
- ii. It is used for showing how to apply modelling techniques for software process development following the agile approach. Sometimes an agile modelling approach helps the developers to get a new idea or compare various alternatives which significantly reduce the complexity of solving problem.
- iii. It is also used for improving the existing system with the modelling activities following the agile modelling approach.

Basically, AM focuses on the effective modelling and documentation. Although AM models are proven by the code, it does not include the programming activities. It also does not include the testing activities but may include the testability of the model. Other activities like project management, system deployment, system operation, or system support is not included in AM. It includes only the software processes; however it can be used with other full fledged processes such as XP, Scrum, DSDM or UP.

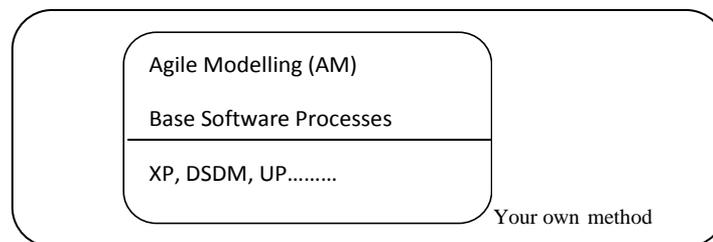


Figure 9: Agile Modelling and Base Software Process.

Figure 9. shows the base software processes such as XP, Scrum, UP or your own personal process which can be tailored with AM. The best part of the AM is that it is possible to pick the best features from different existing software process and can be modelled it using AM to make your own process according to your own necessity. AM

is independent of other processes such as XP or UP, but it plays a significant role in enhancing those processes.

Any person who follows the agile methodology applying the AM practices with its principle and values are agile modellers. An agile developer is who follows the agile approach to software development. Therefore, agile modellers are agile developers but not all the agile developers are not agile modellers.

4. Pitfalls of XP

XP is a lightweight agile methodology with four core values: simplicity, communication, feedback and courage [Beck, 1999a]. Although XP has many interesting practices such as planning game, very short releases and test first coding among others, it is not free of pitfalls. Some of the most common pitfalls from the software point of views are discussed below:

4.1 Requirement

Requirements engineering is the process of specifying requirements by studying stakeholder needs and the process of systematically analyzing and refining those specifications [Jones, 1996]. Specifications are the concise and clear statements that serve as a requirement that the software should satisfy [Macaulay, 1996]. Requirement engineering must include four activities: elicitation, modeling, validation, and verification to produce clear and faultless requirements. Unclear and deficient requirement is one of the biggest causes of software failure [Hofmann, 2001]. According to study done in several hundred organizations by Jones [1996], it was discovered that requirement was deficient in more than 75 percent of organization. Requirements are the mutual agreement and determination of customer needs, user needs, and supplier specifications of software product before it is produced. The requirements define the “what” of a software product [Westfall, 2006] :

- What the software must do?

The answer to this question is the functional requirements.

- What the software must be?

The answer to this question is the non-functional requirements.

- What limitations there are about the choices?

The answer to this question is constrained or limitation of the software product.

Different levels and types of requirement as adopted from Wiegers [2004] are shown in Figure 10:

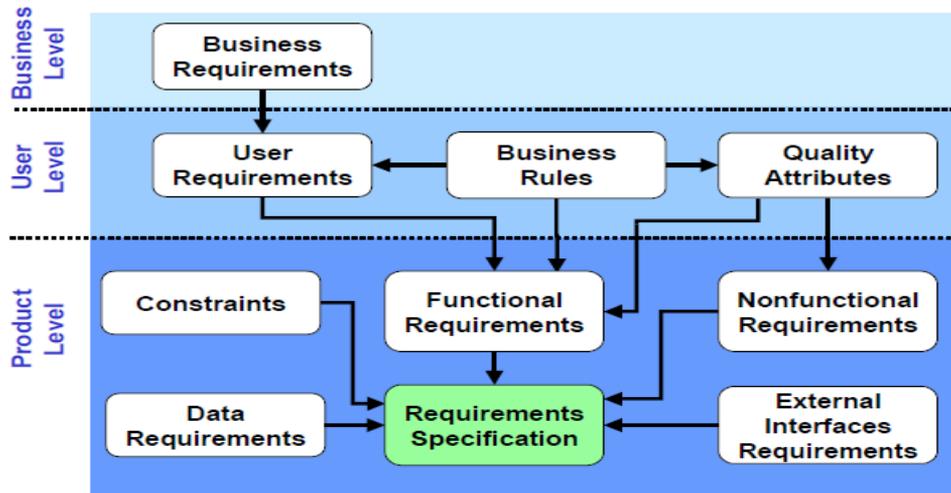


Figure 10: Different levels and type of requirements [Wieggers, 2004].

Figure 10 shows the classification of requirement in different levels and types that helps practitioners to gain better and deeper understanding needed to elicit, analyze, specify, and validate the requirements of software product before development. Business requirements are concerned with the business needs to be addressed by the software product. In general, the goal of business requirements is to clarify the reasons of the software product being developed. User requirements are concerned with the functionality of the software product from the user's perspective. It talks more about the user functionalities of software products. Product requirements are more concerned with the software functionalities to be built into the product to accomplish the overall objectives of user, product and business. [Westfall, 2006]

The requirement process in XP is different than the traditional methodologies. In XP, requirements are the user stories that consist of a few sentences (1-3 sentences) written on an index card which describes the functionality of the customers' values. It serves as the starting point for developers and customers generate more precise detail [Fowler, 2000]. And then the developer decomposes the user story on a card into manageable chunks of tasks recording each task and its status on the card. As there is no analysis of stakeholders and their roles in requirement process, it is very difficult to know the specific requirements of the specific stakeholder.

Information about the requirements of the whole system by a single customer may lead unclear and deficient requirements because single customer does not know all the requirements of the concerned stakeholders. A stakeholder is defined as any group or individual who can affect or is affected by the achievement of the organization's objectives. One of the best solutions to avoid unclear and deficient requirement is to collect use scenarios and perform stakeholder analysis. A use scenario is the

implemented description of techniques that helps to understand the task related activities and also facilitates communication among stakeholders and experts. Stakeholder analysis is an approach for understanding a system by identifying the stakeholders in the system, and assessing their respective interests in, or influence on the system.

Another big problem in XP requirement is that the customer wishes high expectations exaggerating the computer capacity and proposes the more functionalities request and hope that the developers deliver the product in very short time. This usually happens if the customer is unknown about the new technology and available platforms for development. Another major problem in XP is paying less attention towards the changing requirements which leads to project stagnation, modification on finished work and even abandon the finished work. [Li-li et al., 2011]

Modifications to the XP requirements process are reported by many researches and studies. There are various solutions suggested by different studies. But, most of the suggestions are based on the comparative studies. Scenario Based Requirement Engineering (SBRE) practice is proposed in this study.

4.2 Onsite Customer

The customer is supposed to be present on the development site with the developers and has the ability; knowledge and courage make a decision. It is believed that the customer involvement is a key factor for XP project success. However, it is very difficult to implement onsite customer in real practice. In real practice the scope of software development expands to different stakeholders with their own responsibilities. So, what would be the outcome of the development process where requirements, specifications, testing and business decisions are given by the single person representing the respective stakeholder? Another problem is that the present customer representative is often not the end user of the system and the end user is often not capable of making business decisions. [Cao et al., 2004] Multiple customer and Surrogate customer models are proposed as solution to onsite customer practice of XP.

4.3 Pair Programming

Pair Programming (PP) is agile software development practice with two programming working at single work station and one is a driver who writes the code while another is the observer who reviews each line of codes and their roles switches frequently [Williams et al., 2000]. PP is one of the emerging, popular and the most controversial practice in the field of software engineering [Swamidurai & Umphress, 2012]. Some studies have shown that PP is more effective than traditional programming while other

studies have shown that PP is not always practical due lack of resources like small team and also due to lack of developer's interest. Many studies and researches have shown that it is a good practice, but is not true for all cases [Curtis et al., 1988]. In reality, most of the developers do not like to code in pairs, because they are habitual of solo coding [Cao et al., 2004]. One of the practices of XP that draws the ire of XP critics is Pair Programming. The most common criticism is that two developers working together cannot have the same level of maturity and cannot equally contribute to the productivity of the product. However, several studies show Pair Programming is beneficial to traditional programming. The cost of project rises if two developers are assigned to the same tasks at the same time. It is proved statistically that the cost of Pair Programming is 15% higher than traditional programming. It is a hard task to follow the Pair Programming effectively because it depends on the cultivation of personalities within the development team. Another the most common criticism of Pair Programming is that it can be slow process if there raises a lot of disagreement between two developers. But, it can be countered balanced by other practices such as use of common metaphor to describe the problem, simple design, unit testing and coding standard. [Williams et al., 2000] The most common critics of Pair Programming are listed below:

- a. The practice is not realistic in a big organization because developers are working concurrently with many projects at the same time and is also not realistic to small organization because there is always lack of resources like human resources. For example, one developer has to work for many projects at the same time [Swamidurai & Umphress, 2012].
- b. It requires good management system to make sure that the pair working together is more fruitful to the organization than they work separately. It requires efficient and effective evaluation method to measure tangible properties like number of features implement and intangible properties like quality of code [Swamidurai & Umphress, 2012].
- c. The Pair Programming largely depends upon the personal traits of the developer sitting for Pair Programming. A study carried out with 196 software professionals in three countries forming 98 pairs have shown that the personality traits have modest predictive value on Pair Programming performance [Hannay et al., 2010].

Personality traits development training, Distributed Pair Programming (DPP) [Dou et al., 2009] model and Collaborative Adversarial Pair Programming (CAPP) [Swamidurai & Umphress, 2012] model are proposed as alternative to traditional Pair Programming (PP).

5. Addressing Pitfalls through Agile Modelling

Why Agile Modelling approach was used in modellingXP? The answer is very simple; Agile Modelling is a part of XP. It uses many Agile Modelling techniques such as User story, Component Responsibility Collaborator (CRC) cards, models and sketches. There are mainly two primary purposes for using modelling approach. First is to understand and make others understand what is being built and what are the processes involved in it. Second is to analyze the requirement and present detail design of the system. My work is concerned with both of the primary purposes of using modellingapproaches. I have used Agile Modelling for clarifying the necessity and analyzing them in term of agile models. I have used Agile Modelling approach for requirement modelling and Pair Programming modelling; and conceptual modelling approach to onsite customer practice to make them realistic and practical in real XP project.

5.1 Requirement Model

Requirements play significant role to make any project successful. However, unclear and deficient requirements in software development often lead to disappointment with an unreliable product which may even results dangerous accidents. So with unclear and deficient requirements usually create more problem than they solve. One of the major determining factors to make the software development organization successful is how well they understand and manage their requirements. Requirement engineering is the process of developing requirements through an iterative co-operative process of analyzing the problems, documenting the resulting observations in a variety of representation formats and checking the accuracy of the understanding gained [Pohl, 1995]. One of the major problems when dealing with the requirement in XP is that it is very difficult to find someone who can be the real representative of client business [Janeiro, 2001]. Different stakeholders have different interests or perception of business. A single person is not supposed to take decision regarding all the aspects of business. There are always high chances of unclear and deficient requirements collected from a single representative of an organization.

My proposal is to collect use scenarios to get clear and adequate requirements. Use scenarios can be defined as the implemented description of techniques that helps to understand the task related activities and also facilitates communication among stakeholders and experts. The effectiveness of using scenarios in several subjects can work as the capability of simulating thinking. In simple words, scenarios are the representation of the real world and can be generalized for requirement analysis to produce the required models which are familiar to requirement engineers or software

engineers. Figure 11 shows how requirement specifications are related to real world scenarios and how real world scenarios can be used for designing rational models and concept prototypes which helps to extract real requirements from the real world. The best ways of obtaining requirement specifications from usage scenarios are inspection and observation which helps in brainstorming to get the real requirement of the project.

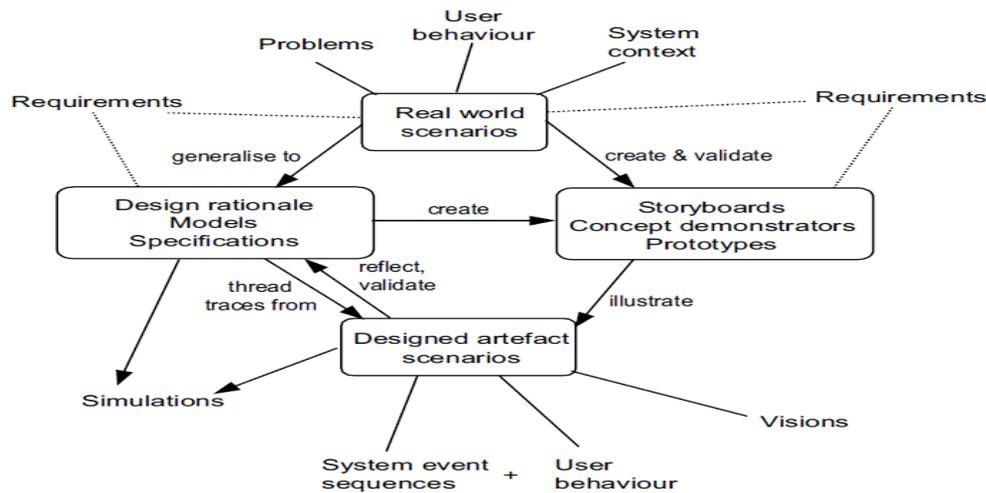


Figure 11: Roles of scenarios and their relationship with requirements [Sutcliffe, 2003].

There are four basic components [Sutcliffe, 2003] in this relationship and they are discussed below:

i. Real World Scenarios

It is the real world of interest that is inspected or observed. Real worlds are always concerned with problems, behaviours and system context. Close inspection and observation helps in brainstorming to derive the real requirement specifications of the system.

ii. Design Rationale-Models Specifications

The real world scenarios can be generalized to rational models with generalized specifications derived from real world scenarios. Possibly, it is a future vision of a designed system with generalized specifications of behavioural and contextual description.

iii. Storyboard-Concept demonstrator prototype

It is a story or example of real world events or grounded theory abstracted from real world experience.

iv. Designed Artefact Scenarios

It is the final designed artefact scenarios derived from the real world scenarios. It is the use case collected from the real world scenario and can be represented in a variety of

formats. It can be sequences of use case diagrams or list of use case requirement specifications.

The major role of scenario is to act as a model to stimulate the designer's imagination. It can help as a guide to support reasoning in the process of designing [Carroll, 2000] but it is not always true. As shown in Figure 12, scenarios play significant starting roles in modelling and contribute in many design processes. Scenario of uses explains system tasks at various stage and context scenarios add necessary information about real world scenario such as the physics system and environment. There are three significant roles of scenarios in requirement and design. [Sutcliffe, 2003]

- i. The first task is to describe the unsatisfactory state present in the current system which should be solved by the new system.
- ii. Vision of operation of new system.
- iii. Describe the behaviours and then representing the users and the existing system.

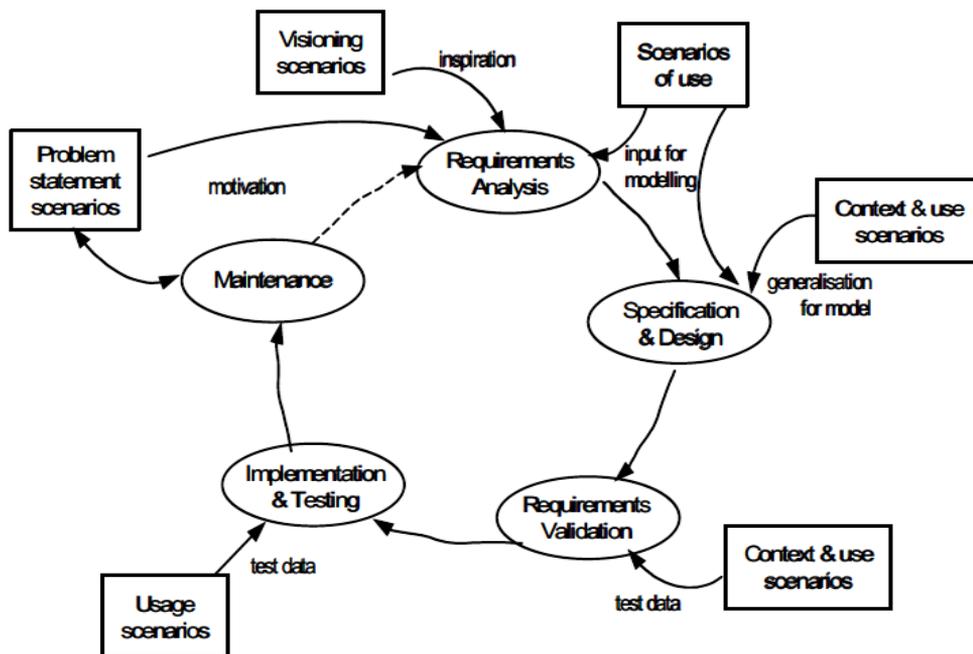


Figure 12: Roles of scenario in requirement and design [Sutcliffe, 2003].

There is always the possibility of eliciting or creating of misuse cases that describes the threats to the system [Alexander, 2002]. The advantages of scenario based requirement are that they provide ground arguments and reasoning in each specification with examples. In scenario based requirement, the patterns of the real world is studied to analyze and then modelled to extract the knowledge. This is quite similar to the

requirement elicitation process which collects the necessary information to extract requirements.

There are two methods in scenario based requirement engineering: (i) ScenIC method, and (ii) SCRAM method [Misra & Kumar, 2005].

i. ScenIC Method

It was proposed by Colin Potts in 1999 and it consists of goals, objective, task, actor and obstacle [Potts, 1999]. The overview of ScenIC method is shown in Figure 13. Scenarios are made up of episode and actions. Man or machines can be actors and goals can be one of the following-achieving states, maintaining states or avoiding states. Obstacles show the successful completion of tasks. In this method, every cycle involves in criticism and inspection of the scenarios that helps to further refine the requirement specifications. General guidelines are provided to format scenario narratives and to identify goals, actions and obstacles. Goals are achieved in episodes and episodes are evaluated with goals achieved. Goals are achieved with the help of system tasks which are carried out by actors. Dependencies are examined among goals, actors, tasks and resources to make sure that all the requirements of the system are met. [Misra & Kumar, 2005]

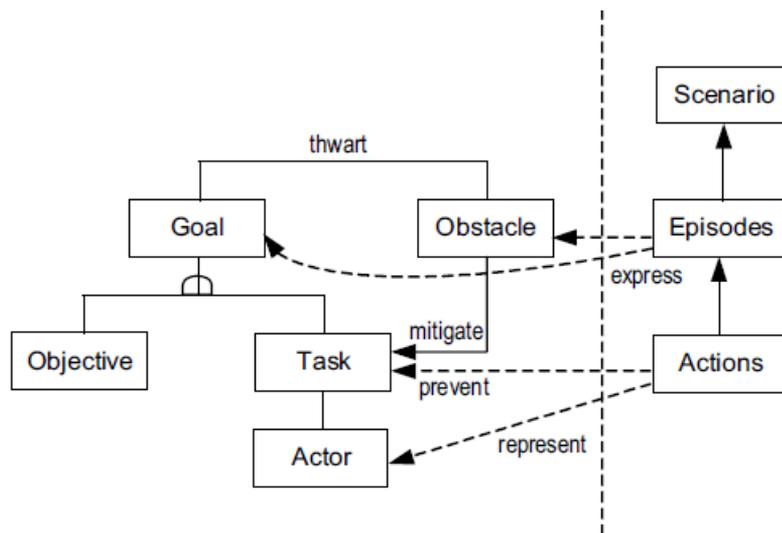


Figure 13: Overview of ScenIC method [Misra & Kumar, 2005].

ii. SCRAM Method

SCRAM stands for Scenario Based Requirement Analysis and this method does not explicitly provide modelling and specification. It works in parallel with software engineering methodology chosen by the practitioner. It is used for requirement elicitation with reasoning about the problem extracted from scenario about use context

[Sutcliffe, 2003]. It is usually done after preliminary design. The general overview of SCRAM is shown in Figure 14:

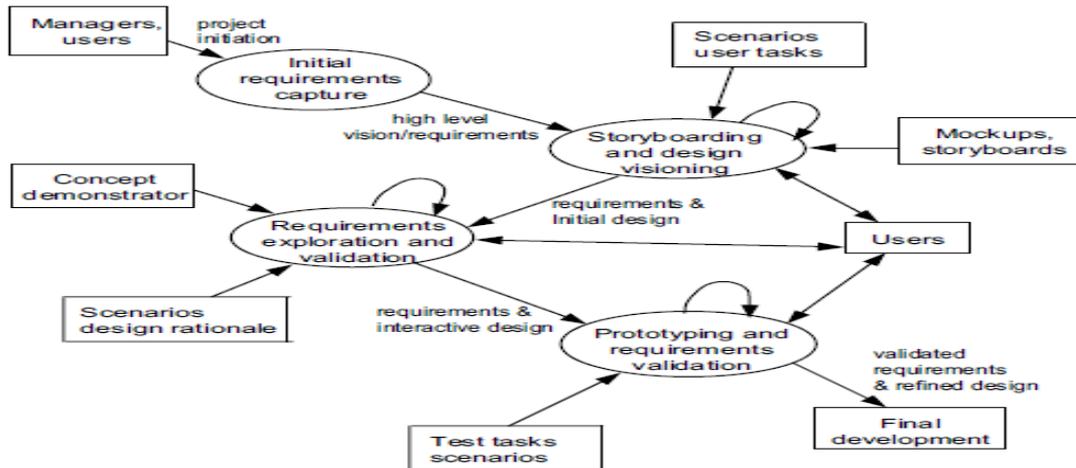


Figure 14: SCRAM overview [Misra & Kumar, 2005].

As shown in Figure 14, SCRAM consists of following four phases [Sutcliffe, 2003]:

i. Initial Requirement Capture and Domain Familiarization

This is the initial stage of SCRAM and initial requirements capturing and domain familiarization is done by conducting conventional interviewing and fact finding techniques. Sufficient information is captured to build first concept demonstrator and it is done after 1-2 client visits.

ii. Storyboarding and Design Visioning

This serves as an early vision for the new system to be designed. The storyboarding explains about the new system in walk-through fashion to get feedback from users.

iii. Requirement Exploration and Validation

Requirement exploration and validation uses the concept demonstrators and early prototyping to come up with more detail design to the users and semi- interactive demonstration are carried out to criticize and validate the requirements.

iv. Prototyping and Requirement Validation

Prototyping and requirement validation is the final iterative process for developing functional prototypes with requirement refinement until prototypes are agreeing to be accepted by the users.

There are some difficulties that should be taken into consideration before following this approach as requirement engineering. The major problem with the approach is that each person has their own individual view of the use context so it is

difficult to filter or generalize the common use context from diverse individual views. Another major problem is volatile human memory. People tend to forget abnormal and rarely occurring problems and the problems that occur frequently and recently are recalled first regardless of their importance and difficulty. There can be a lack of sufficient information to solve the problems encountered.

There are a few scenario based requirement tools that will help to make the process agile. Lexical approaches can be employed for checking and formatting consistency of scenario based requirements [Leite et al., 2000]. CREWS SAVRE version 2.1 built on a Window NT platform using Microsoft Visual C++ and Visual Access supports scenario based requirement engineering with some striking features such as incremental specification of use cases and high-level requirements, automatic scenario generation from use case, description of use cases and scenario of historical data, user walk-through and validation support and so on [Sutcliffe et al., 1998]. It can be effectively used for developing use cases that describe the projected or historical use of the system and then uses a set of algorithms to generate scenarios from the use cases. Furthermore, it can be used for detecting event patterns in scenario by the use of validation frames present in the tool. This helps to provide semi automatic critiques with suggestions for the requirements of a specific scenario. As automated tools are present to facilitate the scenario based requirements, it can be successfully implemented into XP without making it heavy weight methodology.

After requirements are collected from scenario based requirement engineering process, the next step is to identify the stakeholders and perform analysis. As scenario based requirements are focused on collecting and validating the requirements, stakeholders for proposed system needs to be identified and analyzed. Stakeholder identification and analysis are critical first steps to be taken in the participatory planning process and is an area where various approaches can be applied [Renard, 2000]. There are various approaches for identifying stakeholders. Stakeholders might fall under one of the following three categories-internal stakeholders (project member), external stakeholders (not project member but from same organization) and internal to organize but external to both project team and organization. These are the broad classifications of stakeholders. In 2000, Macaulay identified four categories of stakeholder in computer related application domain and they are listed below [Sharp et al., 1999]:

- i. People involved in design and development
- ii. People involved in financial support and are responsible for sale and purchase
- iii. People involved in the introduction and maintenance
- iv. People involved in using the product.

There are many other approaches to identify the stakeholders. From the viewpoint of software and requirement engineering, following are the most appropriate stakeholders staked with the software end product and software development processes:

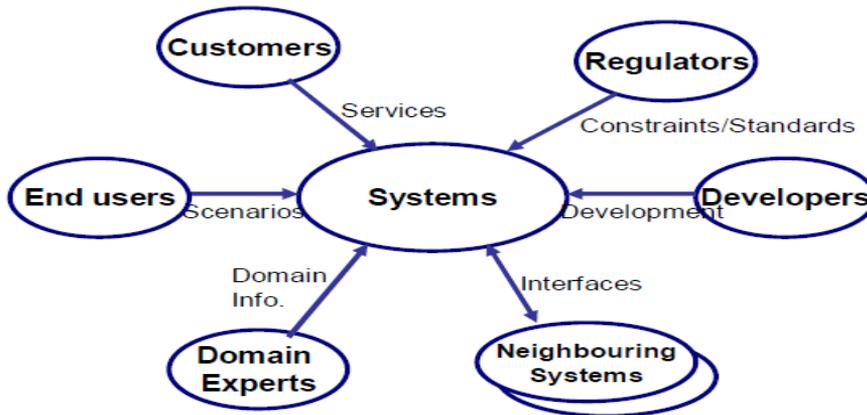


Figure 15: Different types of stakeholders.

Stakeholder analysis is a technique of understanding a system by identifying the stakeholders staked to the system and assessing their relationships, interests and expectation from the system or project. Following are the general steps of stakeholder analysis [de Baar, 2006]:

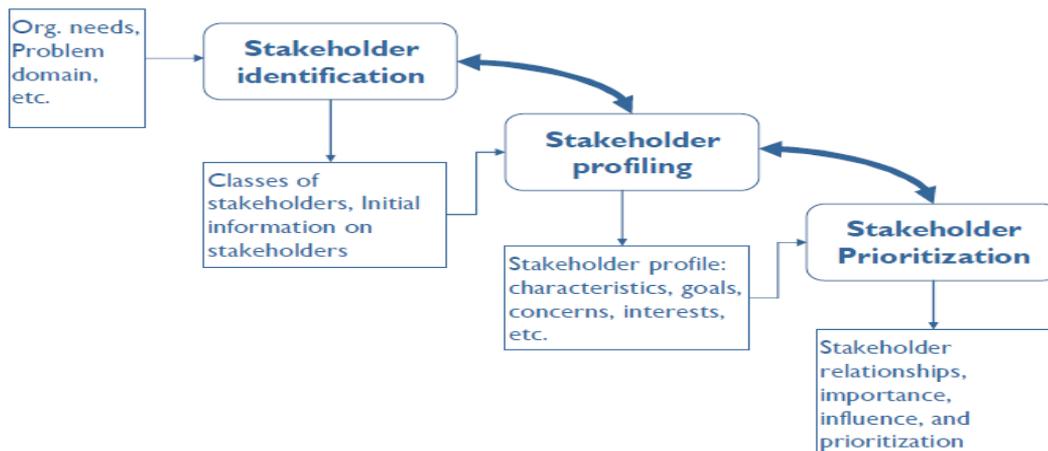


Figure 16: Stakeholder analysis process [de Baar, 2006].

i. Stakeholder Identification

This is the first step of stakeholder analysis process and it is concerned with the question “Who are the stakeholders?” There are various approaches used for identification of stakeholders and some of the most common are:

- a. Checklist
- b. Self selection through documents study
- c. Experts
- d. Identified stakeholders through brainstorming and interviews

ii. Stakeholder Profiling

Stakeholder profiling is concerned with recording the stakeholder concerns and interest to the system. After stakeholders have been identified, the possible interests and concern of identified stakeholders are considered and methods like interview, observation, workshop, document studies can be used for creating profiling of stakeholders. There are various templates available for creating stakeholders' profiling.

iii. Stakeholder prioritization

The third step of stakeholder analysis is to assess the influence and importance of stakeholders so that they can be prioritized according to their influence and importance. Influence is mainly concerned with the power that the stakeholders have over a project. Power over project means the formal control over the decision making process. Influence / importance grids can be used to prioritize the importance and influence of stakeholders. It is shown in Figure 17.

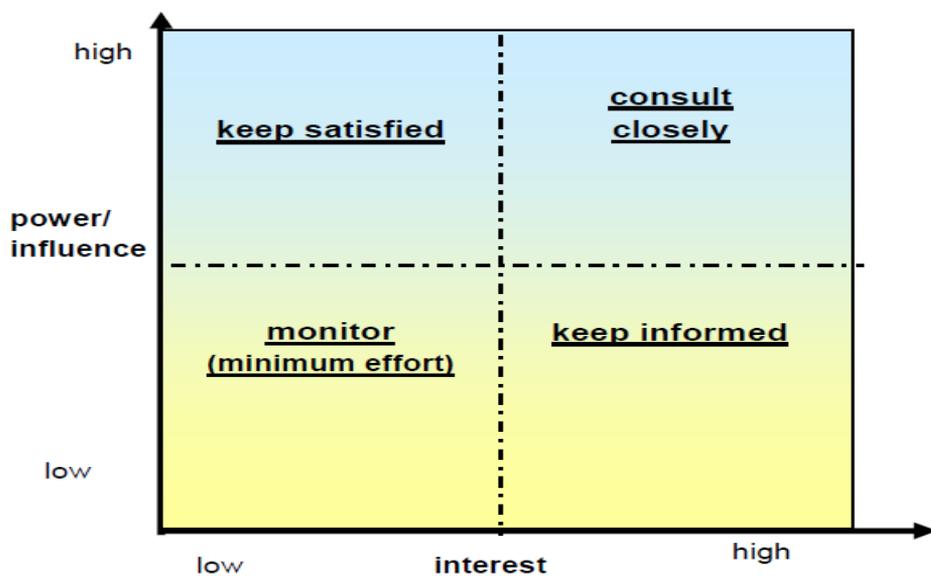


Figure 17: Influence and importance grid.

Why stakeholder analysis is necessary in XP? The requirement engineering process in XP is the most criticized subjects in most of the studies [Li-li et al., 2011][Woit, 2005][Janeiro, 2001]. It is not a difficult process to identify the stakeholders and their roles from scenario based requirement process, but the identified

stakeholders and their roles are not dealt in detail. This helps to make the stakeholder analysis easier as it makes practitioners to identify stakeholders and their role through their inspection and observation. The only task is to create a stakeholders' profile and prioritize their influence and importance based on stakeholder analysis practice. Stakeholder prioritization can be done with the help of Influence and importance grid as shown in Figure 17. Stakeholders are prioritized on the basis of interest and power influence in the grid. Stakeholder involvement helps to avoid the expectation gap between development team and concerned stakeholders. In XP, the requirement is obtained through intensive communication process. This would definitely help to improve the requirement process in XP. Keeping in mind the importance and roles of different stakeholders, a detail user e-story is drafted by XP team. This study demands detail drafted user story and should be available on the web so that it can be referred and documented for future reference. The user e-story contains the detail information about the story description, story number, story priority, story drafted date, risk in story, name of developers responsible for implementation, estimation time, changes in story with date and completed date. This will also help in requirement tracking.

All the above discussed changes are modelled in the release cycle of XP and are shown in Figure 18. The requirement changes are carried out in three stages-collect user scenarios, stakeholder identification and analysis; and detail user e-story.

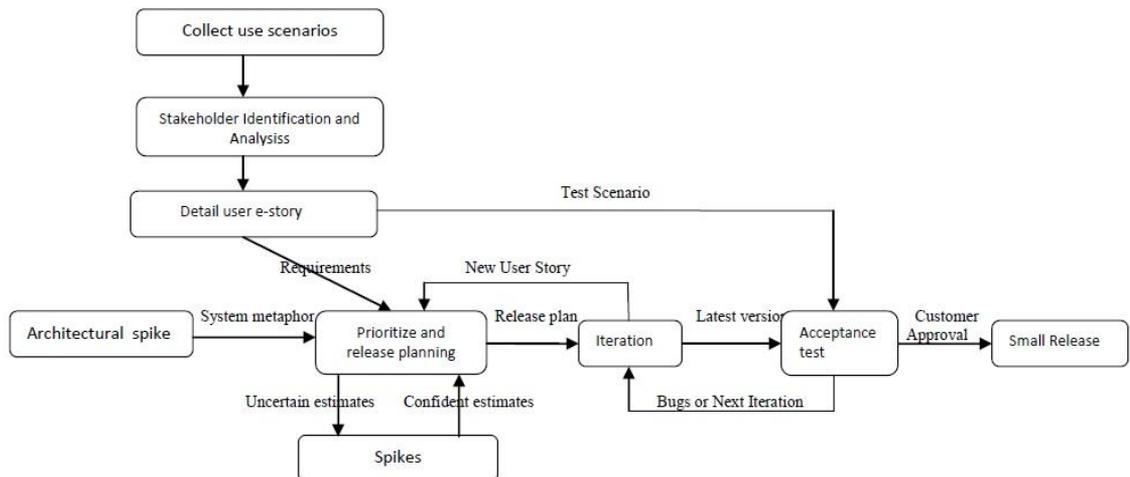


Figure 18: Requirement model in release cycle.

5.2 Onsite Customer Model

Onsite customer is one of the requirements of XP. A customer is not only there to help development team but also he is a part of the development team as well. Onsite customer in XP is responsible for the following roles [Williams et al., 2007]:

- i. To help to develop stories that defines requirements.

- ii. To help to prioritize the features to be implemented in each release.
- iii. To help to develop the acceptance test to make sure that the system meets the desired requirements.
- iv. To make a decision when required.

The roles of onsite customer are very important in XP but the question is not in the roles. The full time availability, domain knowledge of customer and decision making authority are the most criticized points in onsite customer practice in XP. There are very few empirical validated studies on onsite customer. Although the availability of a customer may be valuable, it is not always possible. Wallace et al. [2005] has listed three possible locations of customers: onsite customer, offsite customer and remote customer. Planning in advance is needed if the customer is not present on site. This will help to minimize the risk in the project. It is noted that onsite customers are not only the factor that make XP project successful. There are many other interleaved factors associated with each other to make XP project successful. Beck and Fowler [2000] assumed that the onsite customers are good enough to understand the domain, know how software can provide the business value, and have courage to make decision and willing to take responsibilities for failure and success of the project. Farrell et al. [2002] stated "it is critical to have a high degree of customer involvement in the process". Stephens and Rosenberg states "the trouble with onsite customer done the XP way is that if the onsite customer is a single person, she becomes a single point of failure in an incredibly difficult, stressful, high-profile position of great responsibility". Some studies and researches show that XP onsite customer practice is difficult, costly, impractical and demanding. An empirical controlled XP case study where the customer was present nearly 100% of development time showed that only 21% of his work effort was required to assist the development team [Koskela & Abrahamsson, 2004]. There are many alternative solutions to onsite customer extreme practice of XP. Some of the most common and frequently practiced by practitioners are discussed below.

i. Multiple Customer Representative Model

The general assumption in extreme programming is that an expert customer representative is always remains present to development site but is it is not always possible in the real world [Wallace et al., 2005]. With this technique, single XP development team deals with multiple customers which help to get detail about domain knowledge. The idea is to deal with those customers who have detail and enough information about the domain that the development team is looking for. Multiple customers are contacted or visited on the basis of the priority as set in stakeholder analysis. Multiple customers are not required to be present all the time in the development site. Customers are contacted (or visited or sometimes asked to visit if

necessary) by developers to know about the domain knowledge he/she is working with. This will help with development team to get the right information and decision from the customers. The customers having the highest priority is contacted or visited first and the lowest at last.

ii. Surrogating customer model

Customer involvement is one of the key factors for success of XP projects. However it is very difficult and sometimes even impossible to practice in outsourcing projects. The complexity of the application domain is beyond the expertise and experiences of a single customer in a large scale organization [Cao et al., 2004]. Therefore, the scope of software development is not limited to single customers. Its scope includes a variety of stakeholders who have been identified and analyzed. Development team now includes all the concerned stakeholders. The problem is that it is very tedious and costly to access all the stakeholders and it does not necessarily mean that all the accessible stakeholders are end users of the system.

When the real customers are in accessible especially in a large and complex project, the use of domain expert as a customer would be a reasonable solution to the problem. The act of representing domain expert as the customer is segregating expert as a customer. This practice is very common in outsourcing projects. Surrogating customer model in XP makes outsourcing organization implement XP methodology to develop software.

5.3 Pair Programming Model

Proponents of Pair Programming (PP) claim that PP improves the software development in many perspectives. There are large numbers of studies conducted to prove this claim. However it is one of the extreme practices of XP that has been criticized for a long time. The most common criticism is that two developers working together cannot have the same level of maturity and cannot equally contribute to productivity of same two developers working in parallel [Dick & Zarnett, 2002].

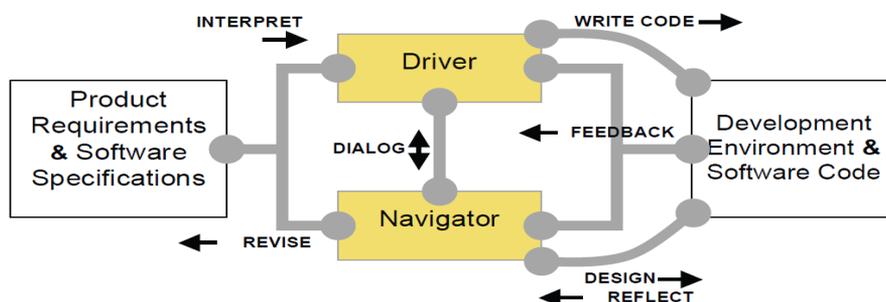


Figure 19: Pair Programming.

As shown in Figure 19, two programmers are involved in Pair Programming (PP) working at single work station with same product requirements and software specifications, and the role of pair programmers changes frequently. Driver is a programmer who writes the code while another is the navigator who reviews each line of codes.

i. Personal Traits Development Training

Effective Pair Programming requires the cultivation of two personalities within the development team. The success of Pair Programming depends upon the personal traits of the persons involved in Pair Programming. So, the successful pairing with good personal traits makes Pair Programming work effectively and efficiently. PP critics claim that the constant disagreement between two developers would slow down the coding task. Dick and Zarnett appointed two senior developers (having development experience of more than 2 years) and four junior developers (have development experience of less than one year) as pair programmers and noted following observations [Dick & Zarnett, 2002].

- a. No dynamic interchange between junior and junior pair as well as senior and junior pair
- b. Project velocity was slow as expected because of a breakdown in interactions. So the pairing was temporarily eliminated after fourth iteration and solo programming was introduced and the developer was responsible for his own work and it worked better.

The possible reasons why Pair Programming did not work in those pairs and concluded that personality traits were lacking in development team and suggested following personal traits needs to be improved for pair programmers [Dick & Zarnett, 2002].

a. Communication

The most important personality trait that is essential for success in Pair Programming is communication. Communication plays important role in every sector. The pair programmer should be able to clearly communicate with each other to discuss and analyze the problem encountered, testing strategy and the bugs found by navigator. There should be no barrier to communicate between driver and navigator in Pair Programming.

b. Comfortable

The navigator and drivers should be comfortable with working environment and with each other. Comfortable pairs can suggest intriguing suggestions and interesting strategies with their knowledge and work of implementing it. Sometime different

working ethic and professional etiquette also affects the comfortable working environment.

c. Confidence

The development team should be confident in their competency and abilities. Confidence in their work such as manipulation of design and code make the confident product. The pair programmers must be confident in their skills to add new features and judge the existing feature.

d. Compromise

The ability to compromise is important personal traits for Pair Programming. Developers who are over confident often lack the compromising traits and are argumentative. Compromise trait helps developer to pick up the best design regardless of its source. The primary idea is to make the pair programmers more flexible for discussion on various suggestions from various approaches and pick up the best one.

Above discussed four personality traits makes the person suitable to Pair Programming. Developers who do not have experience with Pair Programming or feel uncomfortable with the Pair Programming need appear in a training to develop personal traits before pair up. The personal traits training can be provided by the developers who have long and good experiences in PP or by experts.

ii. Improvements in Pair Programming

Following are the proposed models of Pair Programming to improve the XP process. They can be practiced simultaneously.

a. Distributed Pair Programming (DPP) Model

Sitting side by side and having face to face interaction of two programmers in Pair Programming now fails to meet the requirement of global software development. This pointed the necessity of development of platform where developers from different locations can collaborate to solve the same problem. This approach is known to be Distributed Pair Programming (DPP) and is one of the research areas where a lot of experiments are being carried out. DPP is similar to PP in many ways but the developers join virtually to collaborate on the specified tasks from their own computer, keyboard and mouse which help them to work independently. DPP is a derivative of Pair Programming (PP) in a distributed context as emerging development method to support communication and enhance the improvements in PP when developers are geographically apart.

b. Collaborative Adversarial Pair (CAP) Programming Model

Collaborative Adversarial Pair (CAP) Programming is an alternative to Pair Programming and the main objective is to take the merits of Pair Programming while at the same time downplay with its demerits. The main idea is to design together,

construct test and code independently and then test together. An empirical study conducted with twenty six computer science and software engineering senior and graduate at Auburn University in fall 2008 and spring 2009. There were CAP experimental group and PP control group with random distribution of subjects. The subjects were concerned with programming tasks with different level of complexity and used Eclipse and JUnit to perform programming tasks. The result was in favour of CAP and the claim of PP such as reduced time for software development, cost effective, correctness and program quality was supported. [Swamidurai & Umphress, 2012]

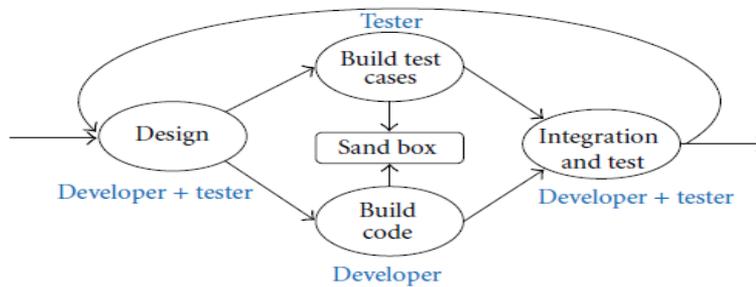


Figure 20: Collaborative Adversarial Pair [Swamidurai & Umphress, 2012].

With the help of agile modelling personal trait development training and collaborative adversarial pair is integrated into XP practice. The agile modelling helps to strengthen the weaknesses of PP that ultimately improves the XP software process. Figure 21 shows the modification on Pair Programming in XP. Personal traits development training and improved Pair Programming are embedded to traditional Pair Programming. The next chapter is concerned with the XP evaluation framework.

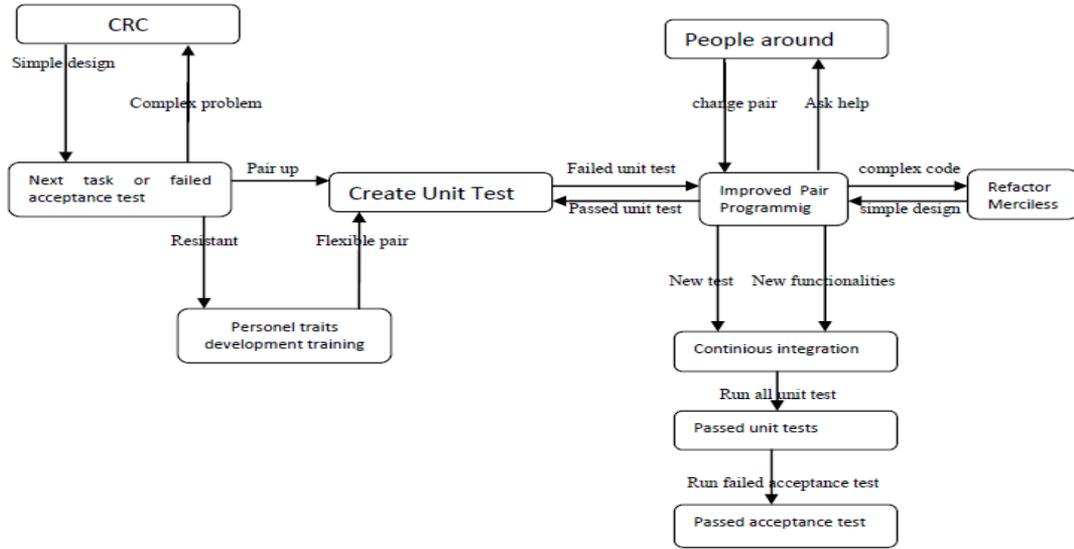


Figure 21: Modification on traditional Pair Programming in XP.

6. Evaluation of XP

As software development processes are used in many domains and come with different shapes and sizes, it is one of the complex human endeavour [Krebs et al., 2011]. We need to measure various aspects of software development methodology and final product to evaluate and understand the effectiveness of the development process. To evaluate the XP, a framework that contains various metrics to capture information about development team, development process, development tools and the final product is designed. This is useful to those organizations which have adapted or willing to adapt XP methodology. The main aim is to build the software process improvement model that can be used for evaluating XP values and practices. Now, the software metrics have become key factors for success of software projects. Measurement is important in software projects because it keeps us involved in it, informs about the current status and provides the guidelines to process further. There are many evaluation frameworks available to evaluate different practices of XP. Usually measurement encompasses of qualitative evaluation and measures in term of numerical values to show the assessment results [Ahmad, 2011]. Karla et al. [2010] proposed a quantitative evaluation framework for agile methodologies and was based on the four postulates of Agile Manifesto. The quantitative evaluation framework based on four postulates of Agile Manifesto cannot evaluate the practices of methods on which it is used. It can only tell about the agility of the agile methods evaluated. The evaluation framework initiated by Willian [2005] is more general agile evaluation framework with no XP focused features. The proposed XP evaluation framework in this study is XP focused and evaluates the XP project, product and practices.

6.1 Meaning of Measurement

According to Fenton and Pfleger [1997], "measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules". An entity can be anything like time, event, commodity, thing, place or person. Measurement is extensively used in most of the production and manufacturing area to estimate costs, calibrate equipment, assess quality and monitor inventories. [Westfall, 2009] Science and engineering disciplines are incomplete without measurement tools and techniques. Why measurements are used? The most general four reasons for measurements are: to characterize, evaluate, predict and improve the existing or proposed system. As shown in Figure 22, attributes of the entity are taken into consideration for the propose of measurement and are assigned with numbers or symbols.

We need to first determine the entity to be measured. For example, a person is selected as an entity to be measured. Once we select the entity, the attributes of the entity must be selected to measure. For example, personality attributes height and weight can be taken into consideration to be measured. Finally the standardized mapping system must be used to express the quantitative measure of the entity. The height of the person is 5.9 and weight of person is 65. This measurement does not give any meaning unless we express with the mapping system like height is 5.9 feet and weight is 65 kg.



Figure 22: Measurement of entity [Westfall, 2005].

6.2 Software Metrics

Software metrics are the integral part of the state of the practice of software engineering. Many customers specify software and quality metrics as a part of their contractual requirements. [Westfall, 2005] As all the attributes of software are difficult to measure, software measurements do not seem to have fully penetrated into industry practices.

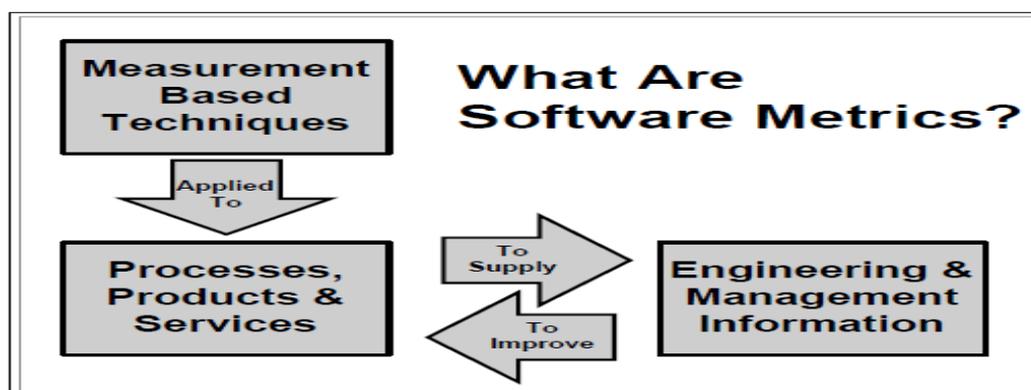


Figure 23: Software Metrics [Westfall, 2005].

A metrics is a quantifiable measurement of software products, process, or project that is directly observed, calculated, or predicted. As shown in Figure 23, software metrics are the measurement based techniques applied to software process, products and services to supply or to improve the engineering and management information. Metrics facilitates to measure the different aspects of an entity that helps us to determine whether or not we are moving towards our specified objective. So, software metrics essentially measure the software product and the processes by which it is developed. They serve as quantifiable indices that determine the current status of the product and the processes by which it is developed. They are useful in predicting outcomes as well as decisions when required. Metrics need to be defined clearly before using it. Following are the elements that should be clearly defined before using metrics. [Ahmad, 2011]

- i. **Metrics Name:** Appropriate name that has something to do with its functionalities should be given.
- ii. **Metrics Description:** Description of what is being measured.
- iii. **Measurement Process:** How metrics is used for measurement?
- iv. **Measurement Frequency:** How often measurement is used?
- v. **Threshold Estimation:** How are thresholds calculated?
- vi. **Current Thresholds:** Current range of values considered normal for metrics.
- vii. **Target Value:** Best possible value of the metrics.
- viii. **Units:** Units of measurement.

6.3 Proposed Evaluation Framework for XP

The measurement of software and software development process is more complicated as compared to the physical measurement system. The measurements in physical systems are rigidly defined and do not require more effort to quantify them. However, the measurements in software engineering are not so rigidly defined as in physical systems and take a lot of effort to quantify them. Software engineers make very difficult and critical decisions based on the result of such measurements. The evaluation framework for extreme programming is basically based on the assessment and evaluation of various project characteristics, extreme programming characteristics, product characteristics and other additional characteristics. The metrics used for assessments and evaluations of XP are designed to be simple, precise, understandable, economical, timely, consistent, accountable, unambiguous, suitable and reliable.

The proposed extreme programming evaluation framework consists of four sections with numbers of subsections. The general block diagram of the proposed XP evaluation framework is shown in Figure 24:

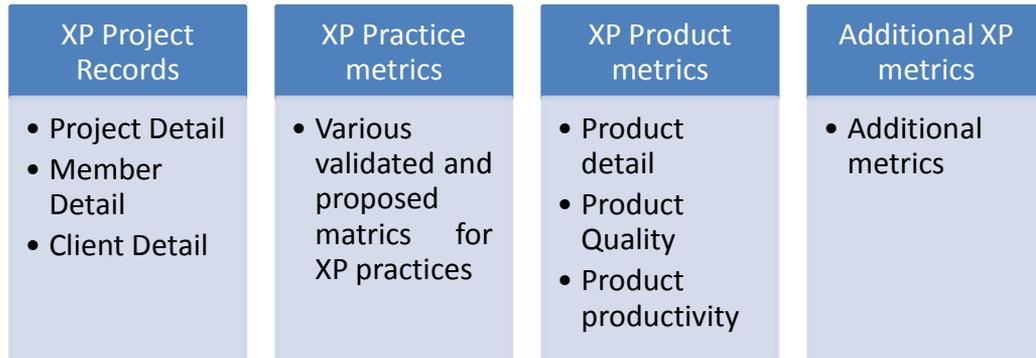


Figure 24: Proposed XP evaluation framework.

Proposed XP evaluation framework design is more specific to extreme programming. It is a collection of some validated and proposed metrics. As illustrated in the figure, proposed XP evaluation framework consists of four sections with some subsections. Subsections of each section are more concerned with both validated and proposed metrics. The first section is Project evaluation which is used for recording and measuring the project and project members' details. The second section is XP practice metrics which contains validated as well as proposed metrics for assessment and evaluation of XP practices used for software development process. The third section is XP product metrics which contains validated as well as proposed metrics for final product assessment and evaluation. The fourth section is Additional XP metrics which contains some validated as well as some proposed metrics for assessment and evaluation of additional information on XP that are not covered in other sessions of proposed XP evaluation framework.

6.3.1. Project Records

Project records are designed in order to evaluate the project and member details. Personnel and team makeup are documented as top risk factors in software development [Boehm, 1991]. Similarly, other factors such as team size, education, work experience and specialization substantially affects the outcome of the project. Following information are recorded in the project records:

6.3.1.1 Project Detail

Project details are recorded in order to assess and evaluate the XP projects in term of cost, schedule and size.

i. Project Name:

It keeps the record of project name.

Record	The name of the project should be relevant. It is either proposed by client or decided by team members working on it. General naming convention can be used. Example: Snake and Ladder for Window Phone 7
--------	--

ii. Project Duration

It quantifies project duration in term of working days and the starting and ending date of the project.

Quantify	The duration of projected is included in terms of days with starting date and ending date of the project. Example: Duration:150 days Starting date: 2013/1/1 Ending date: 2013/4/1
----------	---

iii. Domain

It keeps the record of the domain name of the application built for. Several risks are associated with different domain and important decisions such as selection of languages and database are largely influenced by domain.

Record	Records the domain in which the application is built for. Example: Mobile application
--------	--

iv. Personal Working Hours

This metrics measure the individual working hours contributed to the project. Full time as well as part time workers can be taken into consideration.

Quantify	It quantifies the individual working hours contributed to the project. Example: Sundar Kunwar [Fulltime] 120h
----------	--

v. Time Passed

Time passed metrics measures the overall time spent for project work. The unit of the elapsed may vary from project to project. If the project duration is long, it can be measured in months otherwise it can be measured in days.

Quantify	It quantifies the overall time spent for the project work. Units of measurements can be days or months. Example: Time passed 120 days.
----------	---

vi. Remaining Time

The metrics which measures the time left to complete the project. Time passed deducted from the project duration results remaining time. The units can be days or months depending upon the duration of the project.

Quantify	It quantifies the time left to complete the project. It can be calculated as $\text{Remaining time} = \text{Project duration} - \text{Elapsed time}$ Example: Remaining time 100 days
----------	---

viii. Life

It is the life expectancy of the product. In other words, the expected working period of the final product is the life of that product.

Quantify	It quantifies the life expectancy of the final product. Generally, it is expressed in numbers of years. Example: Life 5 years
----------	--

ix. Project Tools

It records all the project tools used during project work. Project tools play a vital role to make a project successful and timeliness.

Records	It lists all the project tools used in the project. Example: <table border="0" style="width: 100%;"> <tr> <td style="width: 50%;">Project tools</td> <td style="width: 50%;">Purpose</td> </tr> <tr> <td>1. Balsimiq Mockups</td> <td>Design</td> </tr> </table>	Project tools	Purpose	1. Balsimiq Mockups	Design
Project tools	Purpose				
1. Balsimiq Mockups	Design				

6.3.1.2 Member Detail

Member detail maintains the detail records about the current permanent group member working on the project. It is designed to keep the records of following details:

Project Name:						
Project Duration:		From:		To:		
Project Group Name:						
Name	Age	Gender	Education	Specialization	Experiences	Current Position
Mr. Shyam Thapa	30	Male	Master in Computer Science	Mobile and Internet Computing	5 years as Web Developer	Senior Developer

6.3.1.3 Client Detail

It keeps record about the client name, position, organization, address and the proper way of contacting the client.

Project Name:		
Project Duration:	From:	To:
Client Name:		
Client Position:		
Client's Organization		
Client contact address	Email:	Phone:
	Fax:	Mobile:
Preferred way of contact		

6.3.2. XP Practices Metrics

XP has its roots spread in information technology system development where it make the development process more responsive to changing business requirements [Meszaros et al., 2002]. The fourteen principles of XP are: Humanity, Economics, Mutual Benefit, Self Similarity, Improvement, Diversity, Reflection, Flow, Opportunity, Redundancy, Failure, Quality, Baby Steps, and Accepted Responsibility. [Beck and Andres, 2004] However, there are no any measuring means to assess all these practices and principles. Therefore, the proposed XP practice metrics play a vital role to assess the effectiveness of these practices and they are discussed below:

v. Project Velocity

Project velocity is the measure of the time taken (in days) and the number of stories completed in a single iteration. It measures the length of the iteration in days and the tasks completed.

Quantify	<p>It quantifies the duration and the number of stories completed in each iteration.</p> <p>Example:</p> <p>Iteration no. 1</p> <p>Duration: 14 days</p> <p>No of stories completed: 2</p>
----------	--

vi. Automated Unit Tests per User Story

It quantifies the total number of automated unit tests carried out per user story. The main objective of this metrics is to know how many unit tests are created for each user story before they are implemented.

Quantify	<p>It quantifies automated unit test classes per user story.</p> <p>Example:</p> <table style="margin-left: 40px;"> <thead> <tr> <th style="text-align: left;">User Story No.</th> <th style="text-align: left;">Automated unit Tests</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>4</td> </tr> <tr> <td>2</td> <td>2</td> </tr> </tbody> </table>	User Story No.	Automated unit Tests	1	4	2	2
User Story No.	Automated unit Tests						
1	4						
2	2						

vii. Frequency of Automated Unit Test

It shows how often the automated unit tests are carried out. It can be calculated as

$$\text{FAUT} = (\text{total number of unit tests} / \text{total number of classes}) \text{ per user story} * 100\%$$

Quantify	<p>It quantifies the frequency of automated unit test.</p> <p>Example: FAUT=5%</p>
----------	--

viii. Acceptance Tests

It keeps all the necessary information about acceptance tests.

Records	<p>It records the information of acceptance test. It records:</p> <p>How many acceptance tests are written?</p> <p>Who wrote acceptance tests?</p> <p>Who run the acceptance tests?</p> <p>How often acceptances are run?</p> <p>Are all acceptance tests automated?</p>
---------	--

ix. Number of iterations per user story

Implementation of a user story may or may not be fully implemented in iteration. Therefore, it measures the numbers of iterations taken by user story to get fully implemented.

Quantify	<p>It quantifies the number of iterations carried out to implement each user story. It helps to estimate the effort required and the complexity of the user story.</p> <p>Example:</p> <table> <thead> <tr> <th>User story no.</th> <th>Number of iterations</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>4</td> </tr> <tr> <td>2</td> <td>2</td> </tr> </tbody> </table>	User story no.	Number of iterations	1	4	2	2
User story no.	Number of iterations						
1	4						
2	2						

x. Onsite Customer Availability

Onsite is very simple but difficult practice of XP. It is the measure of how often the customer is available on onsite of development. It can be qualified as Full time, Part time and Never.

Qualify	<p>It qualifies the customer availability on the development site with the development team.</p> <p>Example:</p> <p>Customer Availability: Full time</p>
---------	--

xi. Pairing Frequency

In Pair Programming, one programmer is driver who writes code while the other is observer or navigator who reviews the code as it is typed in. The two programmers switch roles frequently. Pairing frequency measures how often the role of driver and navigator changes in Pair Programming.

Quantify	It quantifies the frequency of role changing during Pair Programming in XP. If there is 1 pair and role changes 2 times then pairing frequency = $1/2 * 100\% = 50\%$.
----------	---

6.3.3. XP Product Metrics

XP product metrics are concerned with measuring the product related measurements.

i. Number of Component, Methods and Lines of Codes

Number of components, methods and lines of codes determine the size of the project.

Quantify	It quantifies the requirements, components, methods and lines of code in the project which helps to determine the size of the project. Example: Number of requirements 25 Number of components 50 Number of methods 150 Number of lines of code 10000
----------	--

ii. Productivity Metrics

Halstead proposed the coding productivity metrics and the idea was to determine the productivity from the numbers and types of words used in the program. It is also referred as a token count measure. It can be calculated using the following formula.

[Halstead, 1997]

$$\text{Volume} = \text{length} * \log_2 (\text{vocabulary})$$

$$\text{Where length} = N_1 + N_2$$

$$\text{Vocabulary} = n_1 + n_2$$

n_1 = the number of unique operators

n_2 = the number of unique operands

N_1 = the total number of operators

N_2 = the total number of operands

Quantify	It quantifies the coding productivity of the program. Example: Volume=68
----------	---

iii. Difficulty and Effort Metrics

IBM researchers developed difficult metrics which measure the effort required to understand code and maintain a piece of software. It is calculated as follows.

[Andersson, 1990]

$$\text{Difficulty} = n1/2 * N2/n2$$

$$\text{Effort} = \text{difficulty} * \text{volume}$$

Where,

n1 = the number of unique operators

n2 = the number of unique operands

N2 = the total number of operands

Volume = length * log₂ (vocabulary)

Quantify	<p>It quantifies the level of difficulty and the effort required to understand code and maintain a piece of software.</p> <p>Example: Difficulty=40 Effort=65</p>
----------	---

iv. Defect Removal Effectiveness

Defect Removal Effectiveness (DRE) is defined as the ratio of defects removed during the development phase to defects latent in the product and it is usually expressed in percentage [Kan, 2003].

Quantify	<p>It quantifies the ratio of defects removed during the development phase to defects latent in the product.</p> <p>DRE = defects removed during development phase / defects latent in the product * 100%</p> <p>Example: DRE=40%</p>
----------	---

v. Failure Rate

Failure Rate is the ratio of the number of failures to execution time. It was used by Motorola for finding the purpose of assessing the reliability of the product [Kan, 2003].

Quantify	It quantifies the failure rate and is evaluated as Failure Rate=number of failures/Execution time Example:Failure Rate=5.5
----------	--

vi. Constraint

Constraints are the limitations or restrictions present in the project. It lists all the known present in the system.

Records	It records the constraints present in the project. Example: 1. No provision of automated feedback.
---------	---

6.3.4. XP Additional Metrics

There are many metrics that can be put under additional metrics which can be used for evaluating and measuring various aspects of XP. Metrics can be added according to need and necessity principle. Some of them are discussed below:

i. Customer Problem Metrics

The customer problem metrics is generally expressed in terms of problems per user month (PUM).

PUM = Total problems that customers reported (true defects and non-defect-oriented problems) for a time period /Total number of licenses-months of the software during the period

Quantify	It quantifies the problems of customer and usually expressed in terms PUM. Example: PUM=20
----------	---

ii. Customer Satisfaction Metrics

Customer satisfaction is measured in term of results obtained from customer surveys. The result is analyzed in term of following five levels: Very satisfied, Satisfied, Neutral, Dissatisfied and Very dissatisfied.

Qualify	It qualifies the customer satisfaction in five levels: Very satisfied, Satisfied, Neutral, Dissatisfied and Very dissatisfied.
---------	--

iii. Estimation of Number of Defects

It was first proposed by Jones [1998] for the estimation of the number of defects based on the numbers of functional points of the system. It is calculated as:

Potential Number of Defects=FP^{1.25}

Where FP is the functional points of the system

Quantify	<p>It quantifies the estimates of the number of defects and is expressed as:</p> <p style="text-align: center;">Potential Number of Defects=FP^{1.25}</p> <p style="text-align: center;">Example: Potential Number of Defects=159</p>
----------	---

iv. Halstead Metrics for Effort

It was Halstead [1997] who proposed an effort metrics to determine the effort spent. It

is calculated as:

$$E=V/L$$

where,

E = effort

$$L=N\log_2 n$$

V=Program Volume

N=Program Length

n=Program Vocabulary

Quantify	<p>It quantifies the effort spent in system and it is expressed as</p> <p style="text-align: center;">E=V/L</p> <p style="text-align: center;">Example:34</p>
----------	---

7. Discussion

Several studies have shown that there are enabling as well as limiting factors in extreme practices of XP. A detail study about the rules and practices of XP was carried out through interpretive approach and some enabling and limiting factors were discovered and the most criticized factors such as lightweight requirements, onsite customer and Pair Programming are taken into account to make XP practices more realistic and practical. As Agile Modelling is a part of extreme programming, Agile Modelling is used as modelling approach for two practices: lightweight requirements and Pair Programming and conceptual modelling approach was used for onsite customer practice. An evaluation framework for XP is proposed for evaluating XP projects. The framework is only concerned with XP projects. This introduces several validated and some proposed metrics to evaluate the XP projects. The proposed evaluation framework consists of four sections: XP project records to record project detail, member's detail and client detail, XP Practice metrics to evaluate the practices of XP, XP product metrics to evaluate the XP product and Additional XP metrics to evaluate the additional factors of XP such as defects, efforts, customer satisfaction and so on. Metrics can be added to Additional XP metrics section according to need and demand. Broadly, this study is concerned with following two fields of XP projects:

- i. Modelling the most three criticized practices of XP
- ii. Proposing the evaluation framework for XP

The lightweight requirement is one of the most criticized extreme practices of XP. Several studies that demand the necessity of requirement engineering practices in XP are being carried out. Various approaches are suggested in several studies. This study proposes the scenario based requirements engineering practices for XP with stakeholder analysis to overcome the defects in the requirement practices of XP. It is known fact that the unclear and deficient requirements create more problem than they solve. As very lightweight requirement engineering practices are followed in drafting requirement in XP, there is always danger of drafting unclear and defective requirements. The unclear and defective requirements result the propagation of error throughout the software development cycle. This may result final product with undiscovered errors which is one of the risk factors for customers and software developers. The most common enabling and limiting factor of the requirement process in XP is listed below:

Enabling factors of requirement in XP

- Lightweight process.
- Divide and conquer approach.
- Less effort and time.
- Emphasis on oral communication over written documentation.

Limiting factors of requirement in XP

- It is very difficult to find the real representative of customer business.
- Single person (onsite customer) is responsible for making decisions about the business.
- High chances of unclear and defective requirement collected from a single person.
- Bypassing the requirements engineering practices.

The limiting factors seem to affect more than an enabling factor of the requirement process in XP. Therefore, to eliminate all the limiting factors, new approach for collecting requirements in XP is proposed in this study and the approach is called scenario based requirement engineering process where all the related use cases are collected from the real world working environment. The realistic scenarios are generalized for requirement analysis to get the requirements from it. However, there are some difficulties that should be taken into consideration to follow this approach. The major problem is the diverse individual perception and difficulty in generalizing into common context. Another common problem is the volatile human memory. Human often forgets abnormal and rarely occurring problems and remember the frequently and recently occurring problems regardless of their importance and difficulty. There are some scenario based tools that make the process more organized and simple. As automated tools are present to facilitate the scenario based requirements, it can be successfully implemented into XP without making it heavyweight methodology. For example CREW SAVRE version 2.1 built on Window NT platform supports scenario based requirement engineering such as incremental specification of use cases and high level requirements, automatic scenario generation from use cases, description of use cases and scenario of historical data, user walk-through and validation support among others [Maiden et al., 1998]. With the scenario based approach stakeholder identification and analysis becomes easier and simpler. In most of the cases, it is possible to identify and analyze the stakeholders and their roles from real world scenarios. This makes the requirements stronger and realistic. Stakeholder analysis is performed to understand the system with stakeholders staked to it, their relationships, interests and expectation. It helps to avoid the expectation gap between developers and customers with different interests. As the requirement is obtained through intensive communication process in XP, it will definitely help to improve the requirement process in XP. And then the detail user story is drafted in electronic form that is made available through web pages which will act as written requirement specification in future.

Onsite customer practice is also one of the most criticized extreme practices of XP. Onsite customer is responsible for drafting a user story, sitting together with the whole team. User story acts as requirement specification in XP. He/she is also responsible for user story prioritization that defines the priority of user story to be implemented and development of acceptance tests with developers. It is also believed that onsite customer is courageous enough to make a business decision.

Many studies show that onsite customer practice is effective but unrealistic and impractical. The most common enabling and limiting factors of onsite customer are listed below:

Enabling factors of onsite customer

- Team oriented practices.
- Provides business values.
- Timely decision.
- Bearing responsibilities for failure or success of project.

Limiting factors of onsite customer

- Full time availability.
- Inadequate domain knowledge.
- Decision making authority on single people.

There were not so many studies performed relating onsite customer extreme practices of XP. Out of several alternative solutions to onsite customer, two conceptual models were taken into consideration. First is multiple customer representative models where single customer is replaced by a multiple concerned customers who can provide all the necessary information that the developer is looking for. Second is segregating customer model where the domain experts act as customer in case real customer are inaccessible. Especially, it can be practiced in outsourcing projects.

Pair Programming (PP) is another the most criticized extreme practice of XP. It has been claimed that PP improves software development process in many ways. However, some studies and researches show that two developers working together cannot be productive, economical and chances of delay if developers have strong disagreements on some issues. During my study, I found that there are some basic things to be improved. Personal traits plays significant role in PP. Hence personal traits development training to pair programmers is essential. Two alternative solutions to Pair Programming: Distributed Pair Programming Model and Collaborative Adversarial Pair (CAP) Programming model are proposed in this study.

Enabling factors of Pair Programming

- Collaborative and supportive effort.

- Feel of code ownership.
- Reluctant to interruption-single person can be easily interrupted than a pair.
- Pairs are less likely to go down Gopher Holes and Blind Alleys.
- Two minds are always better than single.

Limiting factors of Pair Programming

- Differences in programming and communication skills.
- Antisocial or anti personalities.
- Perception of cost and time.
- Common schedule and agreement.
- Discourage in pairing.

The personal traits development training is proposed to inexperienced and resistant programmers to help in cultivation of two personalities making them right pair. It helps to improve communication skills, to make more comfortable, confident and comprising which are suitable personal traits for Pair Programming. Two models for improving Pair Programming were proposed. First is Distributed Pair Programming (DPP) when programmers are located geographically apart and the second is a Collaborative Adversarial Pair (CAP) to take the merits and downplay the demerits of PP.

There are some studies that examine the enabling or/and limiting factors of XP. Some of the analytical studies present the alternative solution to limiting factors of XP to improve the XP software process. Table 1 shows the analyzed enabling and limiting factors of User Story of XP. Similarly, Table 2 shows the analyzed enabling and limiting factors of Pair Programming and Table 3 shows the analyzed enabling and limiting factors of onsite customer.

XP Practices	Enabling Factors	Limiting Factors	Remedy/Remedies	References
User Story	Clear vision: The customer has a clear vision of business processes, product requirements and product background.	Deficient Requirement: Customers are not able to give complete requirements to developers. Flood Requirement: Customer has high expectations exaggerating the capacity of computer. Frequent Changes: Frequent changes in requirement will lead stagnation, modify and even abandon the finish work. Negative Influence The contradiction between customers and developers has a negative influence on the demand of high quality.	i. Kano Model Analysis for measuring customer feeling and measuring effects of the product or software quality. ii. High Quality Requirement Analysis to measure the customer wish and developer need. iii. XP Demand Module It is established with Kano Model thinking and High Quality Requirement Analysis to explore the high quality requirements with customer awareness and reduce the misunderstanding in software development process and hidden threats.	[Li-li et al., 2011]
User Story	Not stated	Single Customer The assumption that, in the planning game, the business could be represented by just one customer. Non-functional requirements The lack of consideration of non-functional requirements from the standpoint of the business. Linkage The lack of explicit links between stories and tasks cards to the code Process The lack of a process for producing stories and tasks.	i. A process and a representation are proposed for writing the stories and tasks cards. ii. Also include non functional requirements as user stories. iii. The word should be underlined to show that it has an explicit link with other underlined word. iv. The process is described using SADT diagram to verification and validation.	[Janeiro, 2001]
User story	Rapid Rapid response to changing requirements.	Defects Less predictable, less stable, less reliable and less quality assurance requirements. Informal requirements definition User stories drafted by customer are prioritised, but no formal documentation.	Mapping extreme practices to ISO Process Model	[Erharuyi, 2007]
User story	Unambiguous, Correct, and Understandable Modifiable, Verifiable and Annotated by Relative Importance Complete and Concise Requirements	Not Stated	Not Necessary	[Duncan, 2001]

Table 1: Enabling and Limiting factors of user story found in different studies.

XP Practices	Enabling Factors	Limiting Factors	Remedy/Remedies	References
Pair Programming	<p>Counter Balance The detrimental effects of paired programming are counterbalanced by other XP best practices such as common metaphor, simple design, unit tests, coding standard and the reverse is true.</p>	<p>Productivity Two developers working together cannot equal the productivity of the same two developers working in parallel.</p> <p>Cost It has been statistically shown that paired programming costs approximately 15% more time than traditional programming</p> <p>Personal Characteristics Effective paired programming is difficult to achieve and requires a careful cultivation of personalities within the development team.</p> <p>Dynamic interchange The dynamic interchange of roles is one major problem in PP.</p>	<p>Personalities Traits It was noticed that certain personality traits are beneficial for paired programming.</p> <p>Improvement in interview technique It can be used for ensuring the traits of pair programmers during their interviews.</p>	[Dick & Zarnett, 2002]
Pair Programming	<p>Defects The end defect content is statistically lower.</p> <p>Faster The pair solves the problem fast.</p> <p>Code Review Mistakes can be found during coding.</p> <p>Learning People learn more about the system and software development.</p> <p>Communication It provides an opportunity to improve the communication skills.</p> <p>Understanding Project end with many people understanding the software product.</p>	<p>Cost The development cost for Pair Programming enabling factors is only 15%.</p> <p>Wrong Perception Managers view programmers as a scarce resource, and are reluctant to "waste" such by doubling the number of people needed to develop a piece of code.</p> <p>Tradition Programming has traditionally been taught and practiced as a solitary activity.</p> <p>Reluctant Many experienced programmers are very reluctant to program with another person.</p>	<p>It is only the study of cost and benefits of Pair Programming.</p> <p>No remedy is provided to address its costs.</p>	[Cockburn & Williams, 2002]
Pair Programming	<p>Better code Its premise—that of two people, one computer—is that two people working together on the same task will likely produce better code than one person working individually</p> <p>Benefits Faster software development, higher quality code, reduced overall software development cost, increased productivity, better knowledge transfer, and increased job satisfaction are some benefits of PP.</p>	<p>Time schedule and agreement It requires that the two developers be agreed for the same place at the same time.</p> <p>Management prospective It requires an enlightened management that believes that letting two people work on the same task will result in better software than if they worked separately.</p> <p>Cost The cost of Pair Programming is higher than that of sole programming.</p> <p>Paring Up Novice-expert and expert-expert pairs have not been demonstrated to be effective.</p>	<p>Collaborative Adversarial pair (CAP) programming The main objective is to take the merits of Pair Programming while at the same time downplay with its demerits. The main idea is to design together, construct test and code independently and then test together.</p>	[Swamidurai et al., 2012]

Table 2: Enabling and Limiting factors of Pair Programming found in different studies

XP Practices	Enabling Factors	Limiting Factors	Remedy/Remedies	References
Onsite Customer	<p>Consultation The onsite customer practice offered the team a unique situation to consult with others whenever needed.</p> <p>Demand It was found out that the role of XP onsite customer requires a strong ability to resolve issues rapidly.</p> <p>Work Commitment The development team perceived onsite customer as a strong demonstration of organization's commitment to their work.</p>	<p>Noisy Environment The onsite customer found Pair Programming quite noisy activity and this may have disturbing influence for the customer's real work especially if the customer is accustomed to work alone in a quiet office.</p> <p>Full time onsite customer Onsite customer was nearly 100% present with the development team, but only 21% of his work effort was required to assist the development team in the development.</p>	<p>Noisy environment could be solved by moving the customer's place of work nearby XP project room.</p> <p>This study concluded that full time availability is not necessary in XP. However, the role of the onsite customer is demanding.</p>	[Koskela et al., 2004]
Onsite customer	<p>Participation in the software development processes.</p> <p>Communication bridge among developers, end users and managers Has vital role in drafting user stories and running tests.</p>	<p>Partially onsite customer Management difficulty in frequently changing in requirements.</p> <p>Semantic gap between customer and developer. It is hard to convince management.</p> <p>Non-appointed customers may create problem.</p> <p>Time limitation of the customer.</p> <p>Varying motivation of customer</p> <p>Location of customer</p>	<p>Product Management Team (PMT) can reduce the onsite customer practice's problems effects.</p>	[Mohammadi, 2008]
Onsite customer	<p>Decision Onsite customer has ability, knowledge and courage for decision making.</p>	<p>Difficulty It is difficult to get customer who has knowledge of all domains necessary for development.</p> <p>Scope The scope of software development expands to include a variety of stakeholders.</p> <p>End user An accessible customer is often not the end users of the system.</p>	<p>The onsite customer in FinApp is surrogated by product managers who have direct contacts with customers.</p>	[Cao et al., 2004]

Table 3: Enabling and Limiting factors of onsite customer found in different studies.

During this study, I have noticed following are the most remarkable enabling and limiting factors of user story (lightweight requirement), onsite customer and Pair Programming extreme practices of XP. Alternative solutions are proposed to limiting factors to improve the XP software process. It is shown in Table 4.

Extreme Practices	Enabling factors	Limiting factors	Remedy/Remedies	Remarks
Lightweight Requirements (User story)	<p>Lightweight process</p> <p>Divide and conquer approach</p> <p>Less effort and time</p> <p>Emphasis on oral communication over written documentation.</p>	<p>High chances of unclear and defective requirement collected from a single person.</p> <p>Bypassing the Requirement Engineering Practices.</p>	<p>Requirement Specifications are collected from Scenario Based Requirement Engineering (SBRE) Practices.</p>	<p>SBRE is not so heavyweight method. Processes are simple and easy to practice. However, it is not as simple as user story. Further improvements and modifications are necessary to make the process lightweight.</p>
Onsite customer	<p>Team oriented practices. Provides business values</p> <p>Timely decision</p> <p>Bearing responsibilities for failure or success of project</p>	<p>Full time availability.</p> <p>Inadequate domain knowledge.</p> <p>Decision making authority on single people</p>	<p>Multiple Customers Representative Model</p> <p>Surrogate Customer Model</p>	<p>Multiple customers having adequate domain knowledge are dealt based on their priority. Customers are surrogated by domain experts according to need and necessity.</p>
Pair Programming	<p>Collaborative and supportive effort</p> <p>Feel of code ownership</p> <p>Reluctant to interruption-single person can be easily interrupted than a pair</p> <p>Pairs are less likely to go down Gopher Holes and Blind Alleys.</p> <p>Two minds are always better than single.</p>	<p>Differences in programming and communication skills</p> <p>Antisocial or anti personalities</p> <p>Wrong perception of cost and time</p> <p>Common schedule and agreement</p> <p>Discourage in pairing</p>	<p>Personality traits development trainings to pair resistant.</p> <p>Distributed Pair Programming (DPP) Model.</p> <p>Collaborative Adversarial Pair Programming (CAPP) Model</p>	<p>Training is only provided to those who are found to be pair resistant.</p> <p>DPP is practices when the developers are geographically apart.</p> <p>CAPP is validated model to take the merits and downplay the demerits of Pair Programming.</p>

Table 4: Remarkable Enabling and Limiting factors observed with alternative solutions.

Measurement is necessary in almost all areas to estimate, calibrate, assess and monitor. Science and engineering without measurement tools and techniques cannot be imagined. So, measurement is equally important in software development methodology to evaluate and improve the effectiveness of the development process. The framework that measures or records the information about development team, development process, development tools and final product of XP is proposed with some new and some validated metrics. The proposed framework is more concerned with the XP projects and it measures and records information about XP projects such as project detail, project member's detail and client detail, XP practices such as various new and validated metrics to measure the practices of XP, XP product such as product detail, product quality, product productivity and additional XP metrics such as effort metrics, defect metrics, customer satisfactions and so on. Measurement helps to improve the XP software process. Measurement system makes it possible to consider the weak aspects

of XP and helps to estimate the considerable amount of effort required to be spent on them to improve and strengthen them.

SWOT analysis was done to evaluate the Strengths, Weaknesses, Opportunities, and Threats involved in this work. It identifies the internal and external factors that are favourable and unfavourable to achieve the main aim of the thesis. It includes the following factors [Boyd, 2005]:

Strengths: internal project characteristics that provides advantages

Weaknesses: internal project characteristics that provides disadvantages.

Opportunities: external project characteristics that provides opportunities.

Threats: external project characteristics that causes problems or troubles.

S.No.	Strengths	Weaknesses	Opportunities	Threats
1.	Introduces requirement engineering practice-SBRE which is not heavyweight.	Some sort of documentation oriented practices are proposed	This study is an opportunity to widen the knowledge in the field of agile software development methodologies.	Documentation oriented Chances of misusing cases
2.	Requirement specifications are well understood from real scenario.	Extra effort is always required implementing the new practices.	Proposed alternative solution improves the requirement engineering practices of XP.	Chances of lengthening the project duration since extra effort is always required to implement the proposed practices.
3.	Impractical and unrealistic extreme onsite customer is made practical and realistic.	Difficult to manage multiple customers and find the right surrogate customer.	The proposed practice is realistic and practical in all cases.	Not timely decisions.
4.	CAPP takes merits and downplay demerits of PP.	Oriented towards solo programming	Balance environment to both solo and pair programmers.	Change in real meaning of Pair Programming
5.	Provides XP focussed evaluation framework	All the metices are XP oriented so cannot be used for other methodologies.	Evaluate and assess the XP project to improve XP software process.	Heavyweight and methods focussed.

Table 5: SWOT analysis of the thesis.

8. Conclusion

Agile software development methodologies came into existence to fulfil the changing needs of customers. Agile methodologies are characterized by personal interaction over process, direct communication, short and frequent release, iterative and incremental process, self organization, code crafting and many more. Extreme Programming (XP) is one of the well known agile software development methodologies with sets of values including simplicity, communication, feedback and courage. It is characterized by short development life cycle, incremental planning, and continuous feedback and depends on communication and evolutionary design. The core part of XP consists of a simple set of practices including planning game, small releases, metaphor, simple design, test driven development (TDD), refactoring, Pair Programming (PP), collective ownership, continuous integration, 40 hour week, onsite customer and coding standard. Lightweight processes are introduced in XP with some extreme practices such as lightweight requirement (user story), onsite customer, Pair Programming, test driven development and metaphor among others. The extreme practices and composition variation has made the software development process more complex. Three the most criticized extreme practices-lightweight requirement, onsite customer and Pair Programming were taken into consideration for the study and agile modelling for lightweight requirement and Pair Programming, and conceptual modelling for onsite customer was performed to overcome the pitfalls found during the study. Models need to be validated which can be further studied in future. Another important section of study is the development of XP evaluation framework which uses some new and some validated metrics for evaluating the XP projects, XP practices, XP products and some additional information about XP which can be modified according to changing requirements.

There are many numbers of enabling as well as limiting factors in XP. This study is concerned only with some extreme practices of XP although there are many other extreme practices to be studied. The study concentrates on only three the most criticized practices-lightweight requirement, onsite customer and Pair Programming of XP. In future, further study about other extreme practice can be carried out to refine the practices and make them simple, practicable as well as effective. The study proposes evaluation framework for evaluating XP project with different existing and proposed metrics in order to evaluate it. The evaluation framework consists of enough room to include the desired metrics on specific field of XP project. It is more concerned with the XP project which can not be applied for other methodologies. Software metrics were chosen or porposed to evaluate the XP practices. However, the agility of agile software development methodologies can be somehow affected by the XP evaluation framework.

The proposed XP evaluation framework comprehensive tools for agile software development to evaluate XP practices without imposing excessive burden. With the improvement in XP practices and process, the metrics can also be further modified or added. An active continuation of research is needed for refining and validating the XP evaluation framework to make it possible to implement practically in real projects. This can be done through the international collaboration with software industries to refine and validate the study. After the refinement and validation, it can be used as standard XP evaluation framework in real projects.

The study focused on three extreme practices of XP and development of XP evaluation framework. There is need of similar studies and researches to discover the enabling and limiting factors of other extreme practices and provide the alternative solutions to limiting factors to improve the XP software processes. The XP evaluation framework facilitates XP practitioner for evaluation XP project, XP practices and XP product.

References

- [Abouelela & Benedicenti, 2010] Abouelela, M., & Benedicenti, L. (2010). Bayesian Network Based XP Process Modelling. *International Journal of Software Engineering and Application*, 1(3), 1–15.
- [Agile - Definition, 2013] Agile - Definition and More from the Free Merriam-Webster Dictionary. (2013). Retrieved February 18, 2013, from <http://www.merriam-webster.com/dictionary/agile>
- [Agile Manifesto, 2001] Agile Alliance, The Agile Manifesto. (2001). Retrieved February 18, 2013, from <http://www.agilealliance.org/the-alliance/the-agile-manifesto/>
- [Ahmad, 2011] Ahmad, N. (2011). Software Measurement and Metrics: Role in Effective. *International Journal of Engineering Science and Technology (IJEST)*, 3(1), 671–680.
- [Alexander, 2002] Alexander, I. (2002). Initial Industrial Experience of Misuse Cases in Trade-Off Analysis. *Proceedings of the IEEE Joint International Conference on Requirement Engineering (RE'02)*.
- [Ambler, 2002] Ambler, S. (2002). Agile Modeling; Effective Practices for Extreme Programming and the Unified Process. Retrieved February 7, 2013, from http://www.scribd.com/doc/37142147/Agile-Modeling-Effective-Practices-for-Extreme-Programming-and-the-Unified-Process#outer_page_99
- [Andersson, 1990] Andersson, T. (1990). A Survey on Software Quality Metrics. Åbo Akademi University, Department of Computer Science. Report D-1990-12, December 1990. Retrieved from <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.17.7502>
- [Boehm & Ross, 1989] Boehm, B. W., & Ross, R. (1989). Theory-W Software Project Management: Principles and Examples. *IEEE Transactions on Software Engineering*, 15(7), 902–916.
- [Beck, 1999a] Beck, K. (1999). Embracing Change with Extreme Programming. *IEEE*, 32(10).
- [Beck, 1999b] Beck, K. (1999). *Extreme Programming Explained*. Addison-Wesley Longman Publishing Co., Inc.
- [Berki, 2006] Berki, E. (2006). Examining the Quality of Evaluation Frameworks and Metamodeling Paradigms of IS Development Methodologies. In: E. Duggan & J. Reichgelt (Eds.), *Measuring Information Systems Delivery Quality*, 265-290.

- [Boyd, 2005] Boyd. (2005). December 2005 Newsletter. *SRI Alumni Association*, (December), 1–16.
- [Cao et al., 2004] Cao, L., Kannan, M., Xu P. & Balasubramaniam, R. (2004) How Extreme does Extreme Programming Have to be ? Adapting XP Practices to Large-scale Projects. *Proceedings of the 37th Hawaii International Conference on System Sciences*.
- [Carroll, 2000] Carroll, J. M. (2000). Five reasons for scenario-based design. *Interacting with Computers*, 13(1), 43–60.
- [Cockburn, 2001] Cockburn, A. (2001). *Agile Software Development* (Vol. 3b). The Agile Software Development Series.
- [Cockburn & Williams, 2000] Cockburn, A., & Williams, L. (2000). The Costs and Benefits of Pair Programming. *In Extreme Programming and Flexible Processes in Software Engineering XP2000* (pp. 1–11).
- [Curtis et al., 1988] Curtis, B., Krasner, H., & Iscoe, N. (1988). A field study of the software design process for large systems. *Communications of the ACM*, 31(11), 1268–1287.
- [David, 2010] David, E. (2010). *Research Methods for Political Science: Quantitative and Qualitative Approaches (Google eBook)* (p. 425). M.E. Sharpe. Retrieved from <http://books.google.com/books?id=8PJYznDXQIcC&pgis=1>
- [de Baar, 2006] de Baar, B. (2006). Using Stakeholder Analysis in Software Project Management. Retrieved May 21, 2013, from <http://www.theicpm.com/blog/item/186-using-stakeholder-analysis-in-software-project-management>
- [Dick & Zarnett, 2002] Dick, A. J., & Zarnett, B. (2002). Paired Programming & Personality Traits. *Proceedings of the 2002 Workshops on Database Theory*
- [Dou et al., 2009] Dou, W., Hong, K., & Zhang, X. (2009). A Framework of Distributed Pair Programming System. *IEEE*, 1–4.
- [Duncan, 2001] Duncan, R. (2001). The Quality of Requirements in Extreme Programming. *CrossTalk, Software Defence Engineering*, 19–22.
- [Erharuyi, 2007] Erharuyi, E. (2007). Combining Extreme Programming with ISO 9000 : 2000 to Improve Nigerian Software Development Processes, (March).
- [Fenton & Pfleeger, 1997] Norman E Fenton and Shari Lawrence Pfleeger (1997). *Software Metrics A Rigorous and Practical Approach*. International Thomson Computer Press

- [Fowler, 2000] Fowler, M. (2000). *Planning Extreme Programming Kent Beck* (pp. 1–105). Addison Wesley.
- [Fowler, 2006] Fowler, M. (2006). Continuous Integration. Retrieved May 16, 2013, from <http://martinfowler.com/articles/continuousIntegration.html>
- [Fowler et al., 2002] Fowler, M., Beck, K., Brant, J., & Opdyke, W. (2002). Refactoring: Improving the Design of Existing Code. Retrieved from [http://www.cs.umss.edu.bo/doc/material/mat_gral_137/M.Fowler et al - Refactoring - Improving the Design of Existing.pdf](http://www.cs.umss.edu.bo/doc/material/mat_gral_137/M.Fowler%20et%20al%20-%20Refactoring%20-%20Improving%20the%20Design%20of%20Existing.pdf)
- [Halstead, 1997] Halstead, M. H.(1997). *Elements of Software Science*. Elsevier North-Holland, Inc.
- [Hannay et al., 2010] Hannay, J. E., Arisholm, E., Engvik, H., & Sjøberg, D. I. K. (2010). Effects of Personality on Pair Programming. *IEEE Transactions on Software Engineering*, 36(1), 61–80.
- Hayward, 2000 Hayward, J. (2000). *Introduction to System Dynamics*. University of Glamorgan.
- [Hofmann, 2001] Hofmann, H. F. (2001). Requirements Engineering as a Success Factor in Software Projects. *IEEE Software*, 1(August), 58–66.
- [Janeiro, 2001] Janeiro, R. De. (2001). Extreme Requirements (XR). *Proceedings of the 2001 Requirements Engineering Conference Applied Science* (pp. 1–13).
- [Jones, 1998] Jones, C. (1998). Software Estimation Rules of Thumb. *Proceedings of the 1998 IFPUG conference* (pp. 1–11).
- [Kalermo & Rissanen, 2002] Kalermo, J., & Rissanen, J. (2002). Agile software development in theory and practice. University of Jyväskylä, Department of Computer Science and Information Systems, Report A-2002-8.
- [Kan, 2003] Kan, S. (2003). *Metrics and Models in Software Quality Engineering* (p. 528). Addison-Wesley Professional.
- [Karla et al., 2010] Karla, E., Pablo, C., & Estevez, F. (2010). A Quantitative Framework for the Evaluation of Agile Methodologies. *Journal of Computer Science and Technology*, 10(2), 68–73.
- [Klasky, 2003] Klasky, H. B. (2003). A Study of Software Metrics. *The State University of New Jersey, Graduate School-New Brunswick Rutgers*.
- [Koskela & Abrahamsson, 2004] Koskela, J., & Abrahamsson, P. (2004). Onsite Customer in an XP Project : Empirical Results from a Case Study Related research. *In proceedings of the 2004 EuroSPI*.

- [Krebs et al., 2011] Krebs, W., Ho, C., Williams, L., Layman, L., & Carolina, N. (2011). Rational Unified Process Evaluation Framework Version 1.0. *IBM Corporation*.
- [Kruchten, 2010] Kruchten, P. (2010). Agility and Architecture: Can They Coexist? *IEEE Software*, 27(2), 16–22.
- [Louridas et al., 2008] Louridas, P., Spinellis, D. & Vlachos, V. (2008). Power laws in software. *ACM Trans. Softw. Eng. Methodol.*
- [Leite et al., 2000] Leite, J. C. S. do P., Hadad, G. D. S., Doorn, J. H., & Kaplan, G. N. (2000). A Scenario Construction Process. *Requirements Engineering*, 5(1), 38–61.
- [Li-li et al., 2011] Li-li, Z., Lian-feng, H., & Qin-ying, S. (2011). Research on Requirement for High-quality Model of Extreme Programming. *Proceedings of the 2011 International Conference on Information Management, Innovation Management and Industrial Engineering* (pp. 518–522). IEEE Computer Society.
- [Lumpur, 2009] Lumpur, K. (2009). Review of Agile Methodologies in Software Development. *International Journal of Research and Reviews in Applied Sciences*, 1(1), 1–8.
- [Maiden et al., 1998] Maiden, N. A.M., Minocha, S., Manning, K. & Ryan, M. (1998). CREWS-SAVRE: Systematic Scenario Generation and Use. *Proceeding in the 1998 third international conference on Requirements Engineering, 1998. Proceedings. 1998 Third International Conference* (pp.148,155).
- [Maurer & Martel, 2002] Maurer, F., & Martel, S. (2002). Extreme Programming Rapid Development for Web-Based Applications. *IEEE Internet Computing*.
- [Melis, 2006] Melis, M. (2006). A Software Process Simulation Model of Extreme Programming. Retrieved May 21, 2013, from http://www.diee.unica.it/DRIEI/tesi/18_melis.pdf
- [Meszaros et al., 2002] Meszaros, G., Andrea, J., & Smith, S. (2002). Framework XP – Building Frameworks using XP. *The Pennsylvania State University*.
- [Misra & Kumar, 2005] Misra, S., & Kumar, V. (2005). Goal-oriented or scenario-based requirements engineering technique - what should a practitioner select? *Canadian Conference on Electrical and Computer Engineering, 2005.*, (May), 2288–2292.
- [Mohammadi, 2008] Mohammadi, S. (2008). An analytical survey of “ onsite customer ” practice in Extreme Programming. *International Symposium on Computer Science and its Application*, 1–6.

- [Nawrocki et al. , 2002] Nawrocki, J., Jasinski, M., Walter, B., & Wojciechowski, A. (2002). Extreme programming modified: embrace requirements engineering practices. *Proceedings in the 2002 IEEE Joint International Conference on Requirements Engineering*. IEEE Computer Society.
- [Nawrocki & Wojciechowski, 2001] Nawrocki, J., & Wojciechowski, A. (2001). Experimental Evaluation of Pair Programming. *Proceeding in the 12th European Software Control and Metrics Conference ESCOM*. Shanker Publishing.
- [Pidd, 1994] Pidd, M. (1994). An Introduction to Computer Simulation. *Proceedings in the 26th conference on Winter simulation*. Society for Computer Simulation International.
- [Pohl, 1995] Pohl, K. (1995). Requirements Engineering : An Overview. *Encyclopedia of Computer Science and Technology*, 36, 1–40.
- [Potts, 1999] Potts, C. (1999). ScenIC: a strategy for inquiry-driven requirements determination. *Proceedings IEEE International Symposium on Requirements Engineering (Cat. No.PR00188)*, 58–65.
- [Renard, 2000] Renard, Y. (2000). *Guidelines for Stakeholder Identification and Analysis*. Caribbean Natural Resources Institute.
- [Scrum Alliance, 2013] Scrum Alliance - What Is Scrum? (2013). Retrieved March 14, 2013, from http://www.Scrumalliance.org/pages/what_is_Scrum
- [Sfetsos et al., 2006] Sfetsos, P., Angelis, L., & Stamelos, I. (2006). Investigating the extreme programming system—An empirical study. *Empirical Software Engineering*, 11(2), 269–301.
- [Sharp et al., 1999] Sharp, H., Finkelstein, A., & Gala, G. (1999). Stakeholder Identification in the Requirements Engineering Process. *IEEE*, 387–391.
- [Sutcliffe, 2003] Sutcliffe, A. (2003). Scenario-based requirements engineering. *Proceedings of the 11th IEEE International Conference on Requirements Engineering*, 320–329. IEEE Computer Society.
- [Sutcliffe et al., 1998] Sutcliffe, A. G., Maiden, N., Minocha, S., & Manuel, D. (1998). Supporting scenario-based requirements engineering. *IEEE Transactions on Software Engineering*, 24(12), 1072–1088.
- [Sutherland, 2004] Sutherland, J. (2004). Agile Development : Lessons Learned from the first Scrum. *Cutter Agile Project Management Advisory Service*, 5(20).
- [Swamidurai & Umphress, 2012] Swamidurai, R., & Umphress, D. (2012). Collaborative-Adversarial Pair Programming. *ISRN Software Engineering*, 1–11.

- [Takeuchi and Nonaka, 1986] Takeuchi and Nonaka. (1986). The New New Product Development Game. *Harvard Business Review*.
- [Wiegers, 2004] Wiegers, K. E. (2004). *Software Requirements*. Microsoft Press.
- [Harrison, 1987] Michael A. Harrison, *Introduction to Formal Language Theory*. Addison-Wesley, 1978.
- [Wallace et al., 2005] Wallace, N., Bailey, P., & Ashworth, N. (2005). Managing XP with Multiple or Remote Customers. *Synop Pty Ltd*.
- [Wernick & Hall, 2004] Wernick, P., & Hall, T. (2004). The Impact of Using Pair Programming on System Evolution : a Simulation-Based Study. *IEEE Software Maintenance*, (2003).
- [Westfall, 2006] Westfall, L. (2006). *Software Requirements Engineering: What, Why, Who, When, and How*. The Westfall Team.
- [Westfall, 2009] Westfall, L. (2009). *The Certified Software Quality Engineer Handbook* (p. 640). ASQ Quality Press.
- [Westfall, 2005] Westfall, L. (2005). 12 Steps to Useful Software Metrics. *Proceedings of the Seventeenth Annual Pacific Northwest Software Quality Conference*.
- [Williams et al., 2000] Williams, L., Carolina, N., Kessler, R. R., & Cunningham, W. (2000). Strengthening the Case for Pair Programming, 19–25.
- [Williams et al., 2005] Williams, L., Krebs, W., Layman, L., Antón, A. I. & Abrahamsson, P. (2005). Toward a Framework for Evaluating Extreme Programming. *Empirical Software Engineering*.
- [Williams et al., 2007] Williams, M., Packlick, J., Bellubbi, R., & Coburn, S. (2007). How We Made Onsite Customer Work - An Extreme Success Story. *IEEE Computer Society*, 334–338.
- [Woit, 2005] Woit, D. M. (2005). Requirements interaction management in an extreme programming environment: a case study. *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, 489–494.
- [XP flow Chart, 2013] XP flow Chart. (2013). Retrieved February 18, 2013, from <http://www.extremeprogramming.org/map/project.html>
- [Yong & Zhou, 2009] Yong, Y., & Zhou, B. (2009). Evaluating Extreme Programming Effect through System Dynamics Modeling. *Proceedings of the 2009 International Conference on Computational Intelligence and Software Engineering*, 1–4.