Enabling and Limiting factors in eXtreme Programming (XP) with Evaluation Framework

Sundar Kunwar
sundark@nec.edu.np
Department of Computer Science and Engineering, Nepal Engineering College, Pokhara University
Changunarayan, Bhaktapur, Nepal

Sundar Kunwar *is working as an Assistant Professor in the Department of Computer Science and Engineering at Nepal Engineering College, where he has been since 2007. He has received BE in computer engineering from Tribhuvan University in 2006 and Master in Software Development from University of Tampere, Finland in 2013. His research interest is in improving the agile software development methodologies through agile modeling. In addition he is also interested and has made numerous contributions to robotics projects. He had successfully coordinated robotics competition organized by IOE held on 2067 on theme "Working together for better Nepal" and had stood first position among all the engineering colleges of Nepal.*

Abstract

As agile software development methodologies are used in many domains and come with different shapes and sizes, it is one of the complex human endeavours. Extreme Programming (XP) is one of the well-known agile software development methodologies and is driven by a set of values including simplicity, communication, feedback and courage, but lacks the mechanism to measure these values demanding the evaluation framework to make it measurable and attainable. The main aim of this study is to build the software process improvement model that can be used for evaluating XP values and practices. The proposed XP evaluation framework in this study is XP focused and evaluates the XP project, product and practices. 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. 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.

*[**Keywords** : Agile, eXtreme Programming (XP), evaluation framework, metrics, lightweight requirement, onsite customer, pair programming]*

I. INTRODUCTION

One of the major challenges of agile software development methodologies is to develop a mechanism to measure the various aspects of software development process [1]. Therefore, there is always a need of such measurement mechanism that could quantify the various aspects of software development methodology and final product 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 proposed in this study. This is useful to those organizations which have adapted or willing to adapt XP methodology. 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 [2]. A quantitative evaluation framework was proposed for agile methodologies and was based on the four postulates of Agile Manifesto [1]. 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 William [4] 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.

According to Fenton and Pfleeger [4], "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 [5]. 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 1, attributes of the entity are taken into consideration for the propose of measurement and are assigned with numbers or symbols.

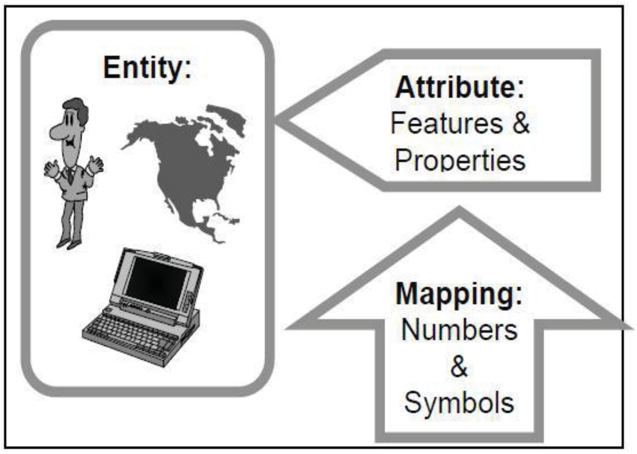

Figure 1: Measurement of entity [5].

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. 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. As all the attributes of software are difficult to measure, software measurements do not seem to have fully penetrated into industry practices A metrics is a quantifiable measurement of software products, process, or project that is directly observed, calculated, or predicted. As shown in Figure 2, software metrics are the measurement based techniques applied to software process, products and services to supply or to improve the engineering and management information.

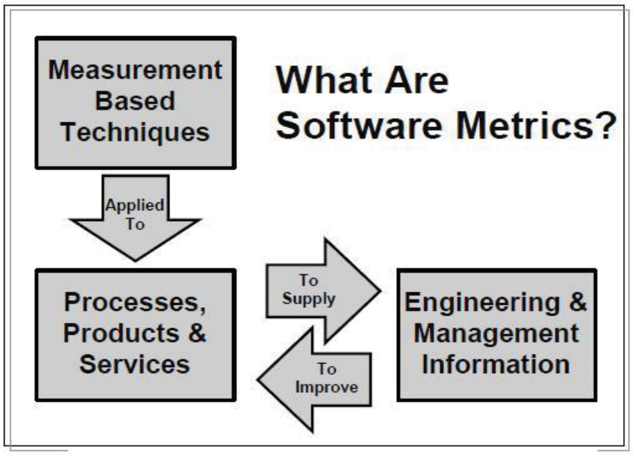

Figure 2: Software Metrics [5]

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. [6]

- **Metrics Name**: Appropriate name that has something to do with its functionalities should be given.
- **Metrics Description**: Description of what is being measured.
- **Measurement Process**: How metrics is used for measurement?
- **Measurement Frequency**: How often measurement is used?
- **Threshold Estimation**: How are thresholds calculated?
- **Current Thresholds**: Current range of values considered normal for metrics.

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.

II. RESEARCH METHODOLOGY

The work is more concerned with the development of evaluation framework with enabling and limiting factors in XP. Most three criticized extreme practices of XP- user stores, pair programming and online customers are mainly taken into account as an initial research framework for discovering enabling and limiting factors and evaluating various aspects of it. An interpretive approach was followed to conduct a literature review. A research can be interpretive if it builds on the assumptions that humans learn about the reality from the meaning they assign to social phenomena such as language, consciousness, shared experiences, publications, tools, and other artefacts [7]. 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. The first component of the hermeneutic cycle is concerned with 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.

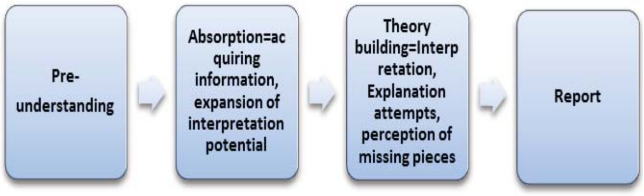
Pre-understanding
Absorption=acquiring information, expansion of interpretation potential
Theory building=Interpretation, Explanation attempts, perception of missing pieces
Report

Figure 3. Hermeneutic Cycle [3]

III. PROPOSED EVALUATION FRAMEWORK FOR XP

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 4:

XP Project Records	XP Practice metrics	XP Product metrics	Additional XP metrics
Project Detail Member Detail Client Detail	Various validated and proposed matrices for XP practices	Product detail Product Quality Product productivity	Additional metrics

Figure 4: 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.

A. 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.

B. XP Practices Metrics

XP has its roots spread in information technology system development where it makes the development process more responsive to changing business requirements [8]. 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 [9]. 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:

1. Sit Together Attendee

Sit together is one of the simplest but most difficult XP practices. XP advocates the entire team members must be present but it is not always possible. Therefore, sit together attendee records the name and of the absentee team member in the meeting.

2. Number of Requirements (User Stories)

The size of the project mainly depends upon the number of user stories which serve as a lightweight requirement to software development process. Simply, it counts the number of user stories in the project.

3. Requirement Complexity

Requirement complexity qualifies how complex is each user story to implement. It can be qualified as low, medium and high.

4. XP Stakeholders

It is used for recording all the concerned stakeholders and their roles in the XP project.

5. 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.

6. 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.

7. Frequency of Automated Unit Test

It shows how often the automated unit tests are carried out. It can be calculated as FAUT= (total number of unit tests/total number of classes) per user story*100%.

8. Acceptance Tests

It keeps all the necessary information about acceptance tests.

9. 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

10. 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.

11. 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.

C. XP Product Metrics

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

1. Number of Component, Methods and Lines of Codes

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

2. 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. [9]

Volume = length*log2 (vocabulary)

Where length = N1 + N2

Vocabulary = n1 + n2

n1 = the number of unique operators

n2 = the number of unique operands

N1 = the total number of operators

N2 = the total number of operands

3. 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. [10]

Difficulty = n1/2*N2/n2

Effort=difficulty*volume

Where,

n1 = the number of unique operators

n2 = the number of unique operands

N2 = the total number of operands

Volume = length*log2 (vocabulary)

4. 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 [11].

5. Constraint

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

D. 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. Some of them are discussed below:

1. 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.

2. Customer Satisfaction Metrics

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

3. Estimation of Number of Defects

It was first proposed by Jones [12] 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

4. Halstead Metrics for Effort

It was Halstead [9] who proposed an effort metrics to determine the effort spent. It is calculated as:

E=V/L

Where,

E = effort

L=NLog2n

V=Program Volume

N=Program Length

n=Program Vocabulary

IV. Enabling and limiting factors in xp

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. The lightweight requirement is one of the most criticized extreme practices of XP. 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. 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 [13]. With the scenario based approach stakeholder identification and analysis becomes easier and simpler. In most of the cases, it is possible to identify and analyse 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.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 analysed enabling and limiting factors of User Story of XP. Similarly, Table 2 shows the analysed enabling and limiting factors of Pair Programming and Table 3 shows the analysed enabling and limiting factors of onsite customer.

XP Practices	Enabling Factors	Limiting Factors	Remedy/Remedies	Ref.
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.	**[14]**
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.	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.	**[15]**
		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.	iv. The process is described using **SADT** diagram to **verification** and **validation.**	
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**	**16]**
User story	Unambiguous, Correct, and Understandable Modifiable, Verifiable and Annotated by Relative Importance Complete and Concise Requirements	**Not Stated**	**Not Necessary**	**[17]**

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

XP Practice	Enabling Factors	Limiting Factors	Remedy/Remedies	Ref.
Pair Programming	**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.	**Productivity** Two developers working together cannot equal the productivity of the same two developers working in parallel. **Cost** It has been statistically shown that paired programming costs approximately 15% more time than traditional programming **Personal Characteristics** Effective paired programming is difficult to achieve and requires a careful cultivation of personalities within the development team. **Dynamic interchange** The dynamic interchange of roles is one major problem in PP.	**Personalities Traits** It was noticed that certain personality traits are beneficial for paired programming. **Improvement in interview technique** It can be used for ensuring the traits of pair programmers during their interviews.	**[18]**
Pair Programming	**Defects** The end defect content is statistically lower. **Faster** The pair solves the problem fast. **Code Review** Mistakes can be found during coding. **Learning** People learn more about the system and software development. **Communication** It provides an opportunity to improve the communication skills. **Understanding** Project end with many people understanding the software product.	**Cost** The development cost for Pair Programming enabling factors is only 15%. **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. **Tradition** Programming has traditionally been taught and practiced as a solitary activity. **Reluctant** Many experienced programmers are very reluctant to program with another person.	It is only the study of cost and benefits of Pair Programming. **No remedy** is provided to address its costs.	**[19]**
Pair Programming	**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 **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.	**Time schedule and agreement** It requires that the two developers be agreed for the same place at the same time. **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. **Cost** The cost of Pair Programming is higher than that of sole programming. **Paring Up** Novice-expert and expert-expert pairs have not been demonstrated to be effective.	**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.	**[20]**

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

XP Practice	Enabling Factors	Limiting Factors	Remedy/Remedies	Ref.
Pair Programming	**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.	**Productivity** Two developers working together cannot equal the productivity of the same two developers working in parallel. **Cost** It has been statistically shown that paired programming costs approximately 15% more time than traditional programming **Personal Characteristics** Effective paired programming is difficult to achieve and requires a careful cultivation of personalities within the development team. **Dynamic interchange** The dynamic interchange of roles is one major problem in PP.	**Personalities Traits** It was noticed that certain personality traits are beneficial for paired programming. **Improvement in interview technique** It can be used for ensuring the traits of pair programmers during their interviews.	**[21]**
Pair Programming	**Defects** The end defect content is statistically lower. **Faster** The pair solves the problem fast. **Code Review** Mistakes can be found during coding. **Learning** People learn more about the system and software development. **Communication** It provides an opportunity to improve the communication skills. **Understanding** Project end with many people understanding the software product.	**Cost** The development cost for Pair Programming enabling factors is only 15%. **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. **Tradition** Programming has traditionally been taught and practiced as a solitary activity. **Reluctant** Many experienced programmers are very reluctant to program with another person.	It is only the study of cost and benefits of Pair Programming. **No remedy** is provided to address its costs.	**[22]**

XP Practice	Enabling Factors	Limiting Factors	Remedy/Remedies	Ref.
Pair Programming	**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 **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.	**Time schedule and agreement** It requires that the two developers be agreed for the same place at the same time. **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. **Cost** The cost of Pair Programming is higher than that of sole programming. **Paring Up** Novice-expert and expert-expert pairs have not been demonstrated to be effective.	**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.	**[23]**

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

During this study, following are the most remarkable enabling and limiting factors noticed and the alternative solutions are proposed to limiting factors to improve the XP software process. It is shown in Table 4.

Extreme Practice	Enabling factors	Limiting factors	Remedy	Remarks
Lightweight Requirements (User story)	**Lightweight process** **Divide and conquer approach** **Less effort and time** Emphasis on **oral communication** over written documentation**.**	High chances **of unclear and defective requirement** collected from a single person. Bypassing the **Requirement Engineering Practices.**	**Requirement Specifications** are collected from **Scenario Based Requirement Engineering (SBRE)** Practices.	**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.
Onsite customer	**Team oriented** practices. Provides **business values** **Timely decision** **Bearing responsibilities** for failure or success of project	**Full time availability.** **Inadequate domain knowledge.** **Decision making authority** on single people	**Multiple Customers Representative Model** **Surrogate Customer Model**	**Multiple customers** having adequate domain knowledge are dealt based on their **priority**. Customers are surrogated by **domain experts** according to need and necessity.

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.	Differences in **programming and communication** skills **Antisocial or anti personalities** Wrong **perception of cost and time** **Common schedule and agreement** **Discourage** in pairing	**Personality traits development trainings to pair resistant.** **Distributed Pair Programming (DPP) Model.** **Collaborative Adversarial Pair Programming (CAPP) Model**	Training is only provided to those who are found to be **pair resistant**. **DPP** is practices when the developers are geographically apart. **CAPP** is validated model to take the merits and downplay the demerits of Pair Programming.

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

V. Conclusion

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 cannot be applied for other methodologies. Software metrics were chosen or proposed 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 is a comprehensive tool 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. 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.

REFERENCES

[1] E. Karla, C. Pablo, & F. Estevez, “A Quantitative Framework for the Evaluation of Agile Methodologies”, Journal of Computer Science and Technology, vol. 10, no.2, pp. 68–73, 2010.

[2] N. Ahmad, “Software Measurement and Metrics: Role in Effective”, International Journal of Engineering Science and Technology (IJEST), vol 3, no. 1, pp. 671–680, 2011.

[3] L. Williams, W. Krebs, L. Layman, A. I. Antón, and P. Abrahamsson, “Toward a Framework for Evaluating Extreme Programming,” Proceedings of the 8th International Conference on Evaluation and Assessment in Software Engineering (EASE ’04), pp. 11–20, 2004

[4] N. Fenton and J. Bieman, Software metrics. Boca Raton, Fla. [u.a.]: CRC Press, 2015.

[5] L. Westfall, The Certified Software Quality Engineer Handbook, ASQ Quality Press, 2009.

[6] N. Ahmad, “Software Measurement and Metrics: Role in Effective”, International Journal of Engineering Science and Technology (IJEST), vol. 3, no. 1, pp. 671–680, 2011.

[7] E David, Research Methods for Political Science: Quantitative and Qualitative Approaches, 2nd ed. Reading, ME Sharpe, 2004, [E-book] Available: http://books.google.com/books?id=8PJYznDXQIcC&pgis=1

[8] G. Meszaros, J. Andrea & S.Smith, “Framework XP – Building Frameworks using XP”, The Pennsylvania State University, Pennsylvania, 2002

[9] Maurice H. Halstead, “Elements of Software Science (Operating and Programming Systems Series”, Elsevier Science Inc., New York, NY, USA, 1978.

[10] T. Andersson, “A Survey on Software Quality Metrics”, Åbo Akademi University, Department of Computer Science,Finland, 1990.

[11] E. Karla, C. Pablo & F. Estevez, “A Quantitative Framework for the Evaluation of Agile Methodologies”, Journal of Computer Science and Technology,vol. 10, no. 2, pp. 68–73, 2010

[12] C. Jones, “Software Estimation Rules of Thumb”, Proceedings of the 1998 IFPUG conference, vol. 10, no. 3, pp. 1–11, 1998.

[13] N.A. Maiden , S. Minocha, K. Manning & M. Ryan, “CREWS-SAVRE: Systematic Scenario Generation and Use”, Proceeding in the 1998 third international conference on Requirements Engineering, vol. 12 no. 5, pp. 148-155, 1998.

[14] Z. Li-li, H. Lian-feng & S. Qin-ying, “Research on Requirement for High-quality Model of Extreme Programming”, Proceedings of the 2011 International Conference on Information Management, Innovation Management and Industrial Engineering, vol. 4 no. 3, pp. 518–522, 2011.

[15] R. De. Janeiro, “Extreme Requirements (XR)”, Proceedings of the 2001 Requirements Engineering Conference Applied Science, pp. 1–13, 2001.

[16] E. Erharuyi, “Combining Extreme Programming with ISO 9000 : 2000 to Improve Nigerian Software Development Processes”, Blekinge Institute of Technology, Sweden, 2007.

[17] R. Duncan, “The Quality of Requirements in Extreme Programming”, Software Defence Engineering, pp. 19–22, 2001.

[18] A. J. Dick, & B. Zarnett, “Paired Programming & Personality Traits”, Proceedings of the 2002 Workshops on Database Theory, 2002.

[19] A. Cockburn, & L. Williams, “The Costs and Benefits of Pair Programming”, In Extreme Programming and Flexible Processes in Software Engineering XP, pp. 1–11, 2000.

[20] R. Swamidurai & D. Umphress, “Collaborative-Adversarial Pair Programming”, ISRN Software Engineering, pp. 1–11, 2012.

[21] A. J. Dick, & B. Zarnett, “Paired Programming & Personality Traits”, Proceedings of the 2002 Workshops on Database Theory, 2002.

[22] A. Cockburn, & L.Williams, “The Costs and Benefits of Pair Programming”, In Extreme Programming and Flexible Processes in Software Engineering XP, pp. 1–11, 2002.

[23] R. Swamidurai & D. Umphress, “Collaborative-Adversarial Pair Programming”, ISRN Software Engineering, vol. 5, no. 8, pp. 1–11, 2012.

* * *