Available onlinewww.ejaet.com

European Journal of Advances in Engineering and Technology, 2018, 5(3): 198-206

Research Article ISSN: 2394 - 658X

Extreme Programming (XP) Simplified

Sundar Kunwar

Department of Computer Science and Engineering, Nepal Engineering College, Bhaktapur, Nepal
sundark@nec.edu.np

ABSTRACT

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 has made the software system more complex and demands the improvements. The main aim of this study is to improve the extreme practices of XP through agile modeling. An interpretive research approach is used to conduct a literature review to develop the agile models. The study deals with modelling the three most criticized and extreme practices (lightweight requirement, Pair Programming and onsite customer) of 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.

Keywords: Agile, extreme Programming (XP), interpretive research, Collaborative Adversarial Pair (CAP) and extreme practices

INTRODUCTION

Software development approaches have been enhancing significantly all the time. It means that software development methodologies are expanding and are becoming increasingly 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. Most of the traditional software development methodologies are very rigid to change. As a result, new software development approaches have evolved as agile methodologies to address the changing requirement of the market. Agile software development methodologies are defined with the ability to respond quickly with changing requirement [1]. 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 [2].

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 1990s, 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 [3]. 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. 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). XP and Scrum are the most commonly used agile software development methodologies.

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 [4]. 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 [4]. This interesting composition of XP is one of the main reasons that make it successful.

It was first Beck and Jeffries who developed the XP as system [3]. 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. The variation in composition and interaction between the values and practices and their feedback in the XP system have made the software system more complex and needs more knowledge to understand each and every common practice of XP [6]. 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 the practices and there are only a few analytical studies of 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. 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.

Extreme Programming is used as research framework to examine the causes why cent percent implementation of XP is not possible. Therefore, an interpretive approach is followed to conduct the literature review and this approach is concerned with the hermeneutic cycle derived from document and literary analysis. The hermeneutic cycle is used for the modeling of extreme programming regarding the most criticized practices of XP. Lightweight requirement, onsite customer and Pair Programming are the three most criticized and extreme practices of XP. Interpretive approach was used for agile modeling to address all the pitfalls of the three extreme practices of XP to make it realistic and practical.

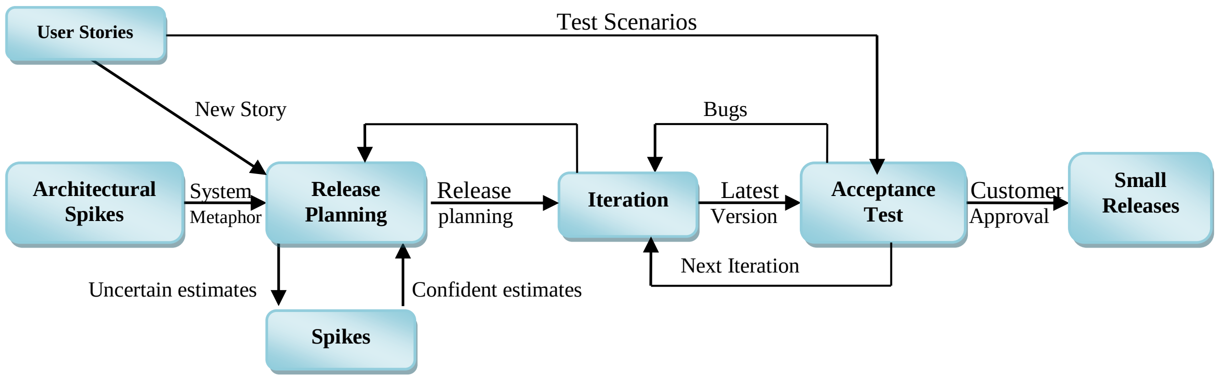

Fig. 1 XP release cycle [5]

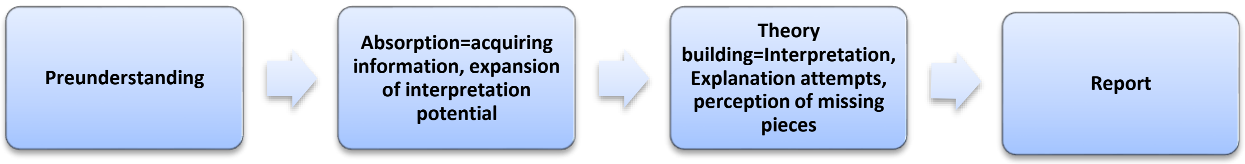
Preunderstanding
Absorption=acquiring information, expansion of interpretation potential
Theory building=Interpretation, Explanation attempts, perception of missing pieces
Report

Fig. 2 Hermeneutic cycle [3]

RESEARCH METHODOLOGY

The aim of this study 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 propose the agile model that best suits XP practices. The work is more concerned with the applicability of XP and is considered Extreme Programming as an initial research framework for explaining 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 meanings 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 2.

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. [3] The same approach of the hermeneutic cycle is used for agile modelling. 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.

AGILE MODELLING (AM)

Agile Modelling is the chaordic, practice based methodology for effective modelling and documenting software based systems [8]. It does not tell about how to build the model, but it tells about how to be effective as modellers. In other word it is not prescriptive process. It is a chaordic because it blends the chaos of simple modelling practices and blends it with the order inherent of software modelling artefacts.

Figure 3 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 need. 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.

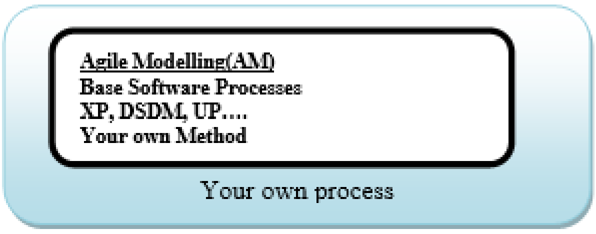

Fig. 3 Agile Modelling and Base Software Process [8]

RESULTS AND DISCUSSION

XP is a lightweight agile methodology with four core values: simplicity, communication, feedback and courage [4]. 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.

Requirement
Requirements engineering is the process of specifying requirements by studying stakeholder needs and the process of analysing and refining those specifications systematically [9]. Specifications are the concise and clear statements that serve as a requirement that the software should satisfy. Requirement engineering must include four activities: elicitation, modelling, validation, and verification to produce clear and faultless requirements. Unclear and deficient requirement is one of the biggest causes of software failure. The process to acquire requirements in XP is different from 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 functionalities of the customers' values. It serves as the starting point for developers and customers generate more precise detail [10]. 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 [10].

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.

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 [11]. Multiple customer and Surrogate customer models are proposed as solution to onsite customer practice of XP.

Pair Programming

Pair Programming (PP) is agile 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 [12]. PP is one of the emerging, popular and the most controversial practice in the field of software engineering [13]. 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 [14]. In reality, most of the developers do not like to code in pairs, because they are habitual of solo coding. 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. However, 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 [12]. Personality traits development training, Distributed Pair Programming (DPP) [14] model and Collaborative Adversarial Pair Programming (CAPP) [15] model is proposed as alternative to traditional Pair Programming (PP) to overcome all the issues discussed.

Addressing Pitfalls through Agile Modelling

Why Agile Modelling approach was used in modelling XP? 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 analyse the requirement and present detail design of the system. This work is concerned with both of the primary purposes of using modelling approaches. Agile Modelling is used for clarifying the necessity and analysing them in term of agile models. Agile Modelling approach is used 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.

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. Therefore, with unclear and deficient requirements create more problems 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 analysing the problems, documenting the resulting observations in a variety of representation formats and checking the accuracy of the understanding gained [16].

To get clear and adequate requirements, collection of use scenarios is proposed in this study. 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 4 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.

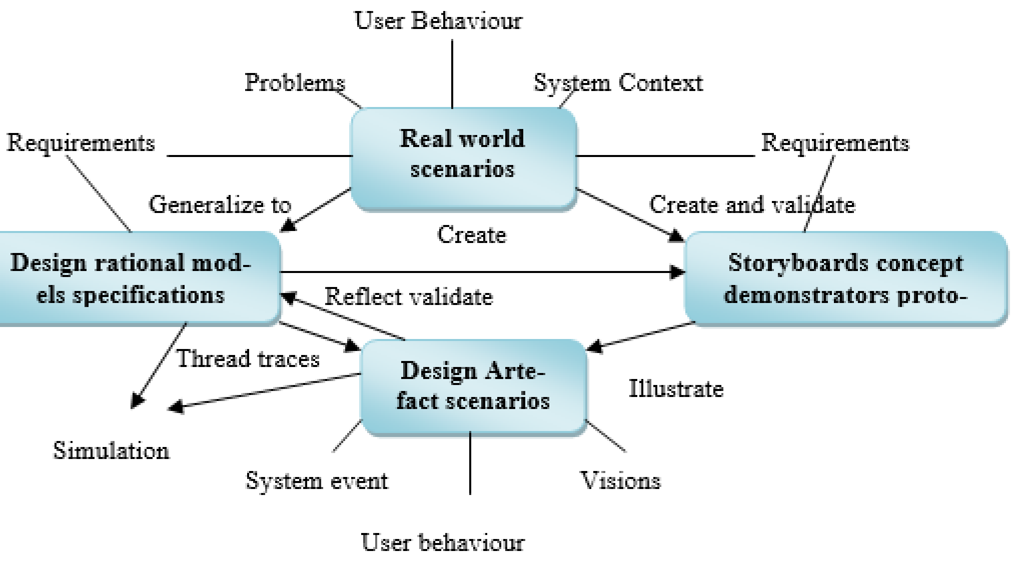

Fig. 4 Roles of scenarios and their relationship with requirements [17]

Real World Scenario 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. The real world scenarios can be generalized to rational models with generalized specifications derived from real world scenarios. Storyboard-Concept demonstrator prototype is a story or example of real world events or grounded theory abstracted from real world experience. Designed Artefact Scenarios 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. There are two methods in scenario based requirement engineering, and they are ScenIC method and SCRAM method [18]. Scenic Method was proposed by Colin Potts in 1999 and it consists of goals, objective, task, actor and obstacle [28]. Scenarios are made up of episode and actions. Man or machines can be actors and goals can 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 [18]. 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 [17]. It is usually done after preliminary design.

After requirements are collected from scenario based requirement engineering process, the next step is to identify the stakeholders and perform analysis. There are many 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 [Fig. 5].

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 Fig. 6 [19].

All the above discussed changes are modelled in the release cycle of XP and are shown in Fig. 7. The requirement changes are carried out in three stages-collect user scenarios, stakeholder identification and analysis; and detail user e-story.

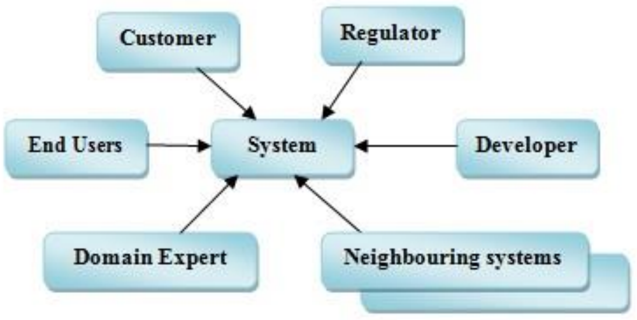

Fig. 5 Different types of stakeholders

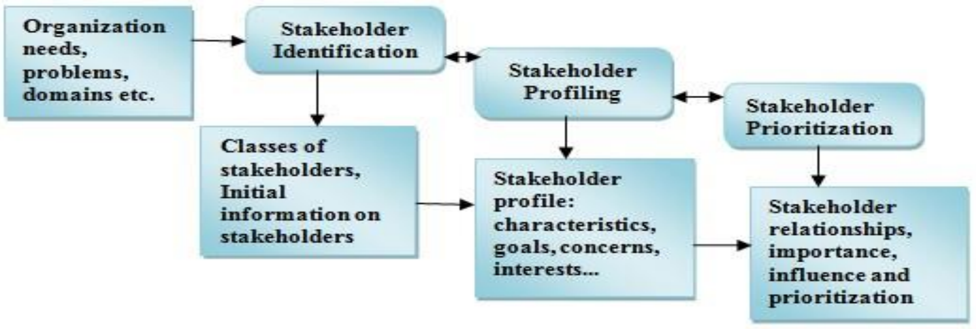
Organization needs, problems, domains etc.
Stakeholder Identification
Stakeholder Profiling
Stakeholder Prioritization
Classes of stakeholders, Initial information on stakeholders
Stakeholder profile: characteristics, goals, concerns, interests...
Stakeholder relationships, importance, influence and prioritization

Fig. 6 Stakeholder analysis process [19]

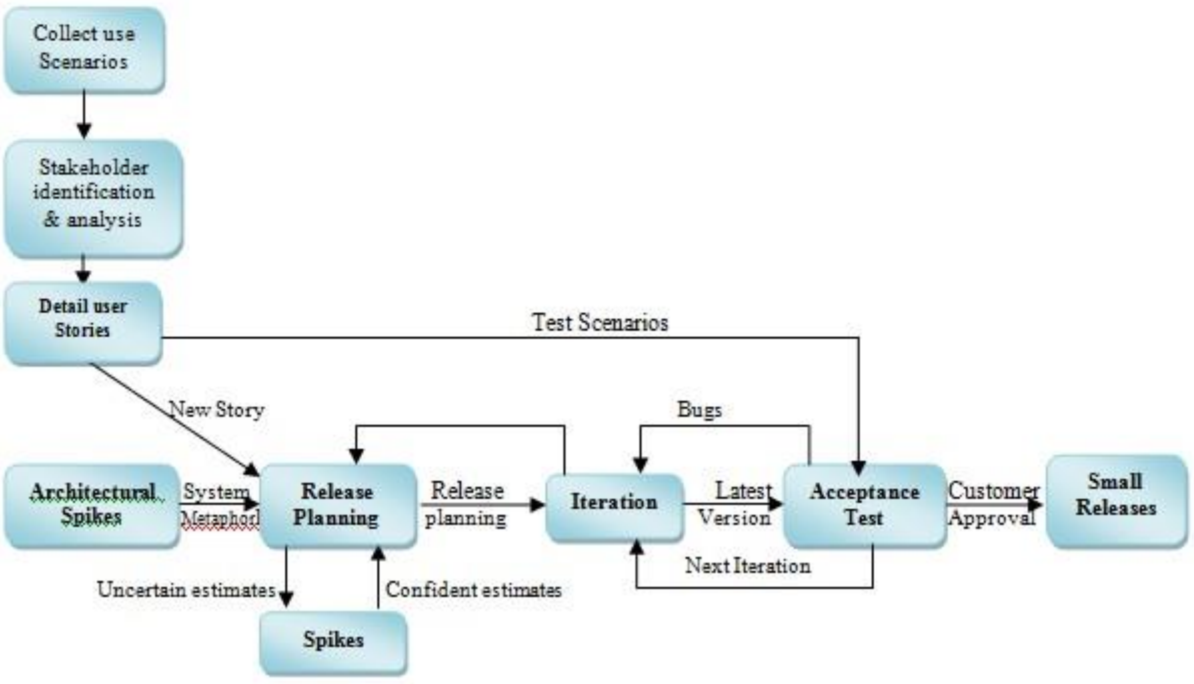

Fig. 7 Requirement model in release cycle

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 assisting in developing stories that defines requirement, prioritizing the features to be implemented in each release, assisting in developing the acceptance test to make sure that the system meets the desired requirement and making a decision when required. [20] 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. [14] 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 [6] [22] 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. 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 [23]. 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.

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 [21]. 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.

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. Therefore, the scope of software development is not limited to single customers. Its scope includes a variety of stakeholders who have been identified and analysed. Development team now includes all the concerned stakeholders. 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.

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 [24].

As shown in Figure 8, 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 programmer’s changes frequently. Driver is a programmer who writes the code while another is the navigator who reviews each line of codes.

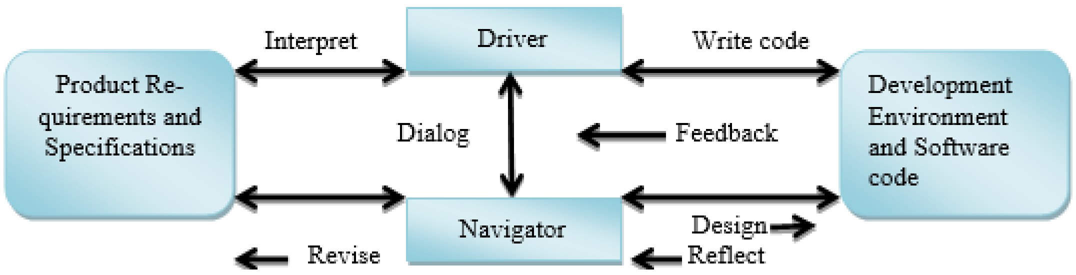
Interpret
Driver
Write code
Product Requirements and Specifications
Dialog
Feedback
Development Environment and Software code
Navigator
Design
Revise
Reflect

Fig. 8 Pair Programming [24]

Personal Trails 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 trails 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 [24].

No dynamic interchange between junior and junior pair as well as senior and junior pair and 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 that personal traits-communication, comfortable, confidence and compromise need to be improved for pair programmers [24].

Improvements in Pair Programming

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

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.

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 [10].

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 9 shows the modification on Pair Programming in XP. Personal trait development training and improved Pair Programming are embedded to traditional Pair Programming.

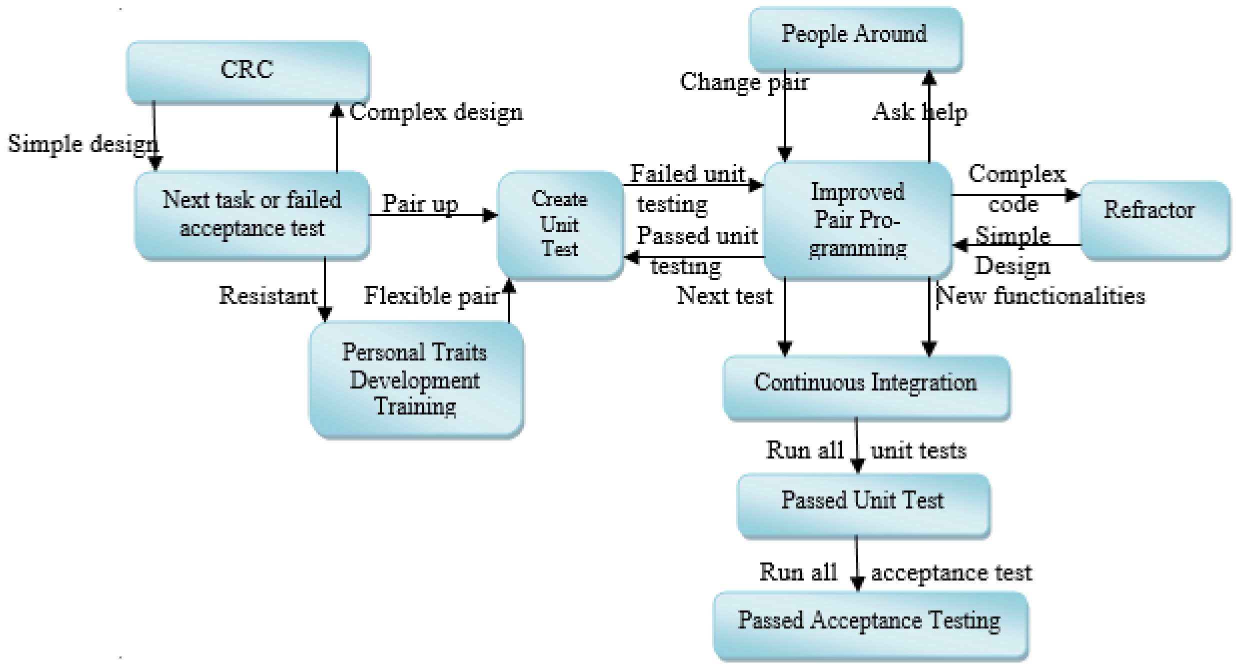

Fig. 9 Modification on traditional Pair Programming in XP

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. 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. 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. The study concluded with requirements in XP should be improved with collecting use cases from scenario based requirement engineering practice followed with stakeholder analysis. Personal development traits, Distributed Pair Programming (DPP) and Collaborative Adversarial Pair (CAP) Programming models are suggested to improve traditional Pair Programming. Surrogate customers and multiple customer models are suggested alternatives to onsite customer. Models need to be validated which can be further studied in the future. The study concentrates on only three the most criticized practices-lightweight requirement, onsite customer and Pair Programming of XP. In the 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] Kuda Nageswara Rao, G Kavita Naidu and Praneeth Chakka, A Study of the Agile Software Development Methods, Applicability and Implications in Industry, *International Journal of Software Engineering and Its Applications,* **2011**, 5(2), 35-4.

[2] P Kruchten, Agility and Architecture: Can They Coexist?, *IEEE Software*, **2010**, 27(2), 16–22.

[3] J Kalermo and J Rissanen, *Agile Software Development in Theory and Practice*, University of Jyväskylä, Finland, **2002**.

[4] K Beck, Embracing Change with Extreme Programming, *IEEE Computer,* **1999**, 32(10), 70-77.

[5] K Beck, XP flow Chart, *Don Wells, Web. http://www.extremeprogramming.org/map/project.html*, **2013**.

[6] K Beck, *Extreme Programming Explained*, 2nd ed., Addison-Wesley, New York, **1999**.

[7] E David, Research Methods for Political Science: Quantitative and Qualitative Approaches, *ME Sharpe, Web. http://books.google.com/books?id=8PJYznDXQIcC&pgis=1,***2010.**

[8] S Ambler, Agile Modeling; Effective Practices for Extreme Programming and the Unified Process, *Wiley Computer Publishing, Web. http://www.scribd.com/doc/37142147/Agile-Modeling-Effective-Practices-forExtreme-Programming-and-the-Unified-Process#outer_page_99,* **2013**

[9] C Jones, Software Estimation Rules of Thumb, *Proceedings of the 1998 IFPUG Conference,* **1998**, 5, 1–11.

[10] M Fowler, *Planning Extreme Programming,* 3rd ed., Addison Wesley, New York, **2002**.

[11] JM Carroll, Five Reasons for Scenario-Based Design, *Interacting with Computers,***2000,** 13(1), 43–60.

[12] L Williams, N Carolina, RR Kessler and W Cunningham, Strengthening the Case for Pair Programming, *IEEE Software*, **2002**, 17 (4),19–25.

[13] R Swamidura and D Umphress, Collaborative-Adversarial Pair Programming, *ISRN Software Engineering*, **2012**, 6(2), 1–11.

[14] B Curtis, H Krasner and N Iscoe, A Field Study of the Software Design Process for Large Systems, *Communications of the ACM*, **1988,** 31(11), 1268–1287.

[15] W Dou, K Hong and X Zhang, A Framework of Distributed Pair Programming System, *IEEE Software,* **2009,** 17(4), 1–4.

[16] K Pohl, Requirements Engineering: An Overview, *Encyclopedia of Computer Science and Technology,* **1995,** 36, 1–40**.**

[17] A Sutcliffe, Scenario-Based Requirements Engineering, *Proceedings of the 11thIEEE International Conference on Requirements Engineering*, **2003,** 12 (10), 320–329.

[18] S Misra and V Kumar, Goal-Oriented or Scenario-Based Requirements Engineering Technique - What Should a Practitioner Select? *Canadian Conference on Electrical and Computer Engineering*, **2005,** 2288–2292.

[19] Bas de Bar, Using Stakeholder Analysis in Software Project Management, *Web. http://www.theicpm.com/blog/item/186-using-stakeholder-analysis-in-software-project-management*,**2006**.

[20] M Williams, J Packlick, R Bellubbi and S Coburn, How We Made Onsite Customer Work - An Extreme Success Story, *IEEE Computer Society*, **2007,** 334–338.

[21] N Wallace, P Bailey and N Ashworth, *Managing XP with Multiple or Remote Customers, Synop Pty Ltd.* ,**2005**.

[22] M Fowler, K Beck, J Brant and W Opdyke, Refactoring: Improving the Design of Existing Code, Web. *http://www.cs.umss.edu.bo/doc/material/mat_gral_137/M.Fowler et al - Refactoring - Improving the Design of Existing.pdf*, **2002**.

[23] J Koskela and P Abrahamsson, Onsite Customer in an XP Project: Empirical Results from a Case Study Related research, *In proceedings of the 2004 EuroSPI*, **2004**.

[24] AJ Dick and B Zarnett, Paired Programming & Personality Traits, *Proceedings of the 2002 Workshops on Database Theory,* **2002**.