I.J.Modern Education and Computer Science, **2013, 2, 41-47**
Published Online February 2013 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2013.02.06

An Assessment of Extreme Programming Based Requirement Engineering Process

Muhammad Khalid
Department of Computing
Shaheed Zulfikar Ali Bhutto Institute of Science and Technology (SZABIST),
Islamabad, Pakistan.
successcodes2009@gmail.com

Sami ul Haq
Department of Computing
Shaheed Zulfikar Ali Bhutto Institute of Science and Technology (SZABIST),
Islamabad, Pakistan.
uomiansame007@gmail.com

Muhammad Naeem Ahmed Khan
Department of Computing
Shaheed Zulfikar Ali Bhutto Institute of Science and Technology (SZABIST),
Islamabad, Pakistan.
mnak2010@gmail.com

Abstract— Comprehensive requirement engineering (RE) process acts as a backbone of any successful project. RE processes are very complex because most of the requirement engineering documentation is written in natural languages, which are less formal and often distract the designers and developers of the system. To streamline different phases of the software lifecycle, first we need to model the requirement document so that we can analyze and integrate the software artifacts. Designers can ensure completeness and consistency of the system by generating models using the requirement documents. In this paper, we have made an attempt to analyze extreme programming based RE approach to understand its utility in the requirement elicitation phase. In this study, different RE process models are evaluated and a comparison of the extreme programming technique is drawn to highlight the merits of the latter technique over the conventional RE techniques.

Index Terms— Requirement Engineering, Extreme Programming, Requirement Elicitation, SDLC

I. Introduction

The field of software development is facing several challenges due to incorporation of nonstandard models and tools in the requirement engineering (RE) phase. Because of this, the failure rate of software projects is increasing rapidly. Proper emphasis on the requirement engineering process is considered as a key to the success of a software project. The objective of this research is to evaluate different requirement engineering process models and do and in-depth study of the eXtreme Programming (XP) approaches. RE is a complex process because most of the RE documents are written in natural languages. The disposition of a natural language always contains ambiguity because it is less formal and often contains confounded information due to multiple denotations of a single word; therefore, its connotation highly depends on the context used. To streamline different phases of software development lifecycle, designers need to model the requirement document in order to integrate the system artifacts which can be easily analyzed. Modeling of the requirement document is necessary to ensure completeness and consistency of the document traceability for maintenance [1,2,3].

Extreme programming is one of the several popular agile processes, and its history dates back to March 6, 1996 when the first extreme programming project was launched. Extreme programming has been reported to be very successful, particularly, in the projects undertaken by the large-sized companies and industries worldwide. According to Jiang [3], RE is a process that constantly reiterates requirement definition, documentation and development, and then it produces confirmed requirements at the end of its process.

Extreme programming puts more emphasis on the teamwork. All the stakeholders, particularly managers, customers, users and developers are all equal partners of a collaborative team. Extreme programming is much like a jigsaw puzzle as it consists of many small pieces — individual pieces do not make sense but when combined together, they depict a complete picture of the project. The amazing aspect of extreme programming is its simple rules; the rules may seem inelegant and perhaps even naive in the first place, but in fact, they are based on sound standards and principles. Extreme programming approach improves a software project in five different

ways: communication, simplicity, feedback, respect and courage. In addition, there are five rules for extreme programming technique: planning, managing, coding, designing and testing. Each of these rules is further subdivided into small chunks of guidelines. In this paper, we only use planning and managing rules of extreme programming.

This paper is structured into four sections. An introduction to extreme programming based requirements engineering processes is provided in this section. The next section summarizes literature review of the current extreme programming and pair programming practices. A critical analysis and comparison of the techniques discussed in Sections are enunciated in Section III. Finally, we conclude in the last section.

II. Insight Into Extreme Programming Practices

Extreme programming is a disciplined approach to software development According to Tchidi and He [4], Six Sigma defines set of practices to improve processes. We can use Six Sigma for RE process model; the design thus evolves will be called Design for Six Sigma, or DFSS for short. However, the paper focuses more on quality improvement than the process improvement.

A. DFSS Methodology

DFSS focuses on process improvement to achieve quality and customer satisfaction. In light of this, DFSS methodology is described as follows:

- Define goals for RE that are consistent with the customer demands.
- Measure and identify the quality characteristics. In the realm of RE, it defines key quality goals that decide upon whether the quality is met or not. If the quality is not met, then it is deemed as a defect.
- Develop designs that meet the defined or required goals and objectives; and analyze the developed design based on the quality characteristics. The designs that do not meet the defined quality characteristics are discarded. If multiple designs meet the quality characteristics, then select the best one among them.
- Optimize the best selected design. This may require constant evaluation of design using simulations or dry runs. The outcome of this exercise will result in creation of a highly optimized design solution.
- Verify the design by designing system prototype, test runs, development and implementation of the software product.

There remains a constant focus on quality in each of these phases. To achieve total quality, several quality management tools may be used. At the same time, each phase is studied for potential requirement changes and associated risks. The identified risks are thoroughly studied and analyzed, and appropriate processes are defined to eliminate them. DFSS focuses on the following key points:

- Continuous focus on quality.
- Defining solutions that ensure adequate transformation of customer requirements.
- Constant improvement of processes. In the realm of RE, these processes are related to requirements' definition.

Fruhling and McDonald [5] highlight a case study for better understanding of extreme programming practices and describe the potential they carry for implementing software systems, particularly those of the government organizations, e.g., the US Military's ability to meet its mission critical requirements demands for increased agility in its IT development processes. The authors have studied the extreme programming process to develop new capability for USSTATCOM's premier knowledge management system and SKI Web, etc. The authors have also reported several lessons learned that may assist practitioners in future implementations of extreme programming practices.

Specifically, the armed forces and law enforcement agencies seek faster and reliable ways of getting critical information and handy access to decision making tools. This type of infrastructure is well-suited for the tasks supporting emergent requirements, e.g., U.S. military requires tools that allow decentralized information to be stored and accessed by all the detachments. Service Oriented Architecture (SOA) supports linear and plan-driven software development methods which are traditionally employed by the military organizations [5]. More recently, agile software development methods like extreme programming [2] and Scrum [6] have emerged as viable solutions to streamline the development process and bring about significant improvements in the processes such as timely delivery of required functionality.

Stapel and Lubke [7] report that the extreme programming turns the conventional software process sideways. Rather than planning, analyzing, and designing for the far-flung future, the practices of extreme programming in the software design should be benefited instead. The study emphasizes for inclusion of extreme programming course in the curriculum for graduate students and setting up computer laboratories equipped with extreme programming tools.

There are twelve extreme programming practices — planning game, small releases, metaphor, simple design, testing, refactoring, pair programming, collective ownership, continuous integration, 40-hour per week working, on-site customer and coding standards (as shown in Figure 1).

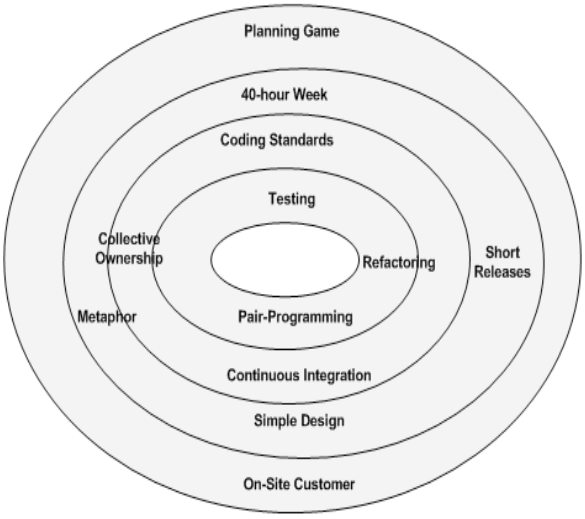

Figure 1. Extreme Programming Practices [7].

B. PLANNING GAME:

The game is defined as a meeting that occurs at least once for each iteration — typically once a week. The planning process is divided into two parts: release planning and iteration planning. *Release Planning* focuses on ascertaining what requirements are included in which near-term releases. Both, customers and developers are part of this process. Release planning consists of three phases.

- *Exploration Phase*: In this phase, the customer provides a short list of high-value requirements for the system. These requirements are noted down on user story cards.
- *Commitment Phase*: The commitment of customers and developers to the functionality that needs to be included into the system along with the date of next release falls within the purview of this phase.
- *Steering Phase*: In this phase, the development plan can be adjusted by adding new requirements or changing/removing the existing requirements.

Iteration Planning outlines plans for streamlining the activities and tasks of developers. Customers are not involved in this process. Iteration planning also consists of the same three phases as described below:

- *Exploration Phase*: In this phase, requirements are converted into different tasks, and these tasks are recorded on the task cards.
- *Commitment Phase*: In this phase, the tasks are assigned to the programmers and the expected time for completion of each task is estimated.
- *Steering Phase*: In this phase, the tasks are performed/executed and the end results are harmonized with the original user requirements.

Small releases though cannot eliminate risks, but their functionality and timely availability can result in the following advantages:

- Customers can benefit from the system functionality in advance without waiting for release of a final version.
- Customers can get an additional mechanism for influencing the future direction of the project, e.g., by changing priorities or by adding or removing features.

A small release typically takes at least three to four months. However, an extremely short release may take only a day.

- ***Metaphor*** is a means of communication among team members and customers. Metaphor adds communication value to agile programming. Metaphor serves two purposes: communication and contributing to the team members in development of software architecture.
- ***Simple design*** requires that programmers should adopt the "simple is the best" approach to software designing. Whenever a new piece of code is written, the developers should ask themselves, 'is there a simpler way to produce the same functionality?' Refactoring should be used to make complex code simpler.
- ***Testing:*** Extreme programming has got an impetus from agile methodology that emphasizes much on testing as it eliminates inherited risks in software projects. Extreme programming helps developers in producing software that is on time, under budget and possesses a higher quality level.

Extreme programming puts the practice of software testing in the spotlight of application development. Testing is a highly specialized activity that came as an afterthought when complex and code-intensive projects rush towards completion. In today's world of escalating quality expectations, testing is a key component of the development process.

Extreme programming accelerates testing as it requires the entire development team to embrace testing. In fact, testing is so critical to the extreme programming methodology that programmers are required to write automated tests before they start writing the software codes. However, there has been a distinctive deficiency of instructions specific to testing and clear understanding how these instructions relate to extreme programming.

- ***Refactoring:*** The continuous improvements in design of the code make it easier to work with. This is in total contrast to what typically happens; little refactoring and a great deal of attention paid to suitably adding new features. Extending and maintaining the code becomes easier by employing refactoring continuously.
- ***Pair Programming:*** Pair programming generally involves switching partners for each assignment. However, team members are often reluctant to switch as they claim that only their contemporary peer's weekly schedule is compatible with them.

- ***Collective code ownership:*** It means that all the team members own code responsibility. Pair programming technique is generally used in this practice. It boosts the development speed more as if an error occurs in the development phase then everyone owns responsibility to correct the error.
- ***Continuous integration:*** Regular and continuous integration ensures that all the team members have access to the latest code.

In contrast with the independent software development, extreme programming offers a magnitude increase in productivity. In addition, the combined understanding of the system by multiple people leads to improvements in the design. Further, the maintenance of this system is simplified to a great extent. According to Schummer and Lukosch [8], distributed pair programming is an agile software development methodology where two programmers located at different geographic locations jointly work using a collaborative real time editor. The key difference between pair programming and distributed pair programming is that the programmers in the latter technique are located at different geographic locations, and various communication means are required to be made available for practicing this methodology. The distributed extreme programming, especially distributed pair programming, is destined to failure unless proper tools are used that support social practices.

- ***Communication:*** In pair programming, programmers work in close proximity where they can easily interact with each other, seek quick help and guidance from others and even can get an impetus from each other's body language and non-verbal communiqué. Whereas, for a distributed pair programming, the programmers have to entirely rely on the verbal and written communication.
- ***Coordination:*** Coordination issue is the main demerit of distributed pair programming. In pair programming, one programmer (called driver) writes code and the other programmer (known as observer or navigator) analyses and tests the code, and provides necessary feedback and guidance. However, the proper coordination arrangements help produce better code that has fewer bugs. However, in distributed pair programming, coordination often suffers from time delays due to detached locations of programmers.
- ***Coding:*** Pair programming carries another advantage of effective code writing over distributed pair programming. In the former technique, an observer or navigator can instantly and constantly review the code; whereas, the lack of instant communication inflicts a disadvantage in the latter technique. However, this disadvantage can be countered by employing good tools that provide real time access to the code.
- ***Teaching:*** One of the advantages of pair programming is the ability to learn from each other through cross questioning and discussion.
- ***Testing:*** Pair programming allows the observer to test the code while it is being written. For distributed pair programming to exploit this advantage, the editor has to provide this ability to the programmers.

According to Murphy *et al.* [9], extreme programming focuses on early releases and improving quality of software. Distance learning approach to teaching extreme programming is, in fact, the study of different tenants of distributed pair programming.

According to Sato *et al.* [10], the use of software metrics helps software developers to access their code and foresee any potential risk or design flaw. Over the years, different code metrics and methodologies have been defined that helped industry produce better quality software. Industry experts have used different code metrics on different types of projects to demonstrate the effectiveness of different software development methodologies. For example, agile projects focus on quality and speedy releases. This methodology innately makes software more robust and less error prone.

C. Metrics for Extreme Programming

The following metrics is used in extreme programming:

- ***XP Radar Chart:*** XP Radar charts show a comparison of different projects. Projects that use agile practices at the early stage exhibit better performance rate.
- ***Lines of Code:*** Projects developed through agile practices have less lines of code.
- ***Cyclomatic complexity:*** This metric measures the amount of decision logic in each module. The projects developed using agile practices exhibit better cyclomatic complexity metric.
- ***Weighted methods per class:*** This metric measures the complexity of classes on the basis of methods per class. The projects that use agile practices have lesser measurements for this metric as compared to other methodologies.

The effective requirement engineering model is essential for success of any software development project [11]. The effective requirement engineering entails different phases. These phases, when properly executed in the projects, result in great degree of success. A brief description of the phases is as under:

- *Requirement elicitation:* This phase includes requirement gathering.
- *Requirement analysis and negotiation:* The gathered raw requirements are rigorously analyzed in this phase and are negotiated with the stakeholders.

- *Requirement specification:* In this phase, a well-defined document is produced in light of the analyzed requirements.
- *System modeling:* Based on the requirement specification, a conceptualized model of the system is built in this phase.
- *Requirements validation:* Once the requirements are defined and documented, the stakeholders verify and validate the requirements. This phase is also called requirements signoff phase. Once the requirements are validated, the system is ready to be developed.
- *Requirements management:* During this phase, the requirements are tracked for any changes and dependencies. Often the dependencies between requirements play an important role for success of the project. This phase is an on-going process and keeps the project requirements on track.

III. Literature Review and Critical Evaluation

Fruhling and McDonald [5] highlight a case study for better understanding of extreme programming and describe how much this approach has potential to be implemented in the government organizations. Lu and Lu *et al*. [1] proposed a model-based object-oriented approach to RE by using OOP OOA, OOD concepts. Lu and Chang [2] present a requirement editor, called MOR editor that supports the objectization and modeling of requirement engineering. MOR editor is a tool designed to assist the processes and of requirement documents, which can benefit requirement engineer to objectize requirement artifacts, link-related requirement artifacts, and construct a consistent and traceable formal model for RE. Mishra and Mishra [12] present the application of combination of RE techniques for a real life complex project (i.e., supply chain management) with higher requirements volatility developed in a small-scale software development organization. Jiang [3] defines RE process as a formal description method and proposes a Requirements Engineering Process Meta-model (REPM). REPM is a simple and unified method for describing different types of processes, in which different RE tools are applied. Stapel and Lubke [7] highlight issues to address when designing an extreme programming course. Conboy [6] highlights developer characteristics for effective agile method. Min and Cheng [13] highlight extreme programming practices and time scheduling interface method.

Solemon and Sahibuddin [14] propose RE process improvement model using Capability Maturity Model Integration (CMMI). Pavanasam and Subramaniam [15] proposed a membrane computing model for software requirement. Tchidi and He [4] used Six Sigma technique and proposed a RE process model based on DFSS. Pandey *et al.* [11] proposed RE process model to produce quality requirements for software development. Schummer and Lukosch [8] discussed distributed extreme programming especially distributed pair programming and argue that project may be destined to failure unless proper tools are used that support social practices. Murphy [9] highlights extreme programming approach and challenges. Sato *et al.* [10] highlight agile software development practices. The summary of the critical analysis of the literature reviewed during the course of this study is provided in Table-I.

Table I. Critical Analysis

Ref#	Technique Used	Key Points	Advantages	Limitations
[1]	UML, Model based Object oriented approach, XML based unified model.	A Model-based object-oriented approach to requirement engineering.	Requirements integration with the artifacts of other phases can be effectively improved.	Identification of the relationships between MOORM and MORE elements to XUM models is not clear.
[2]	Model driven architecture, OOA, OOD, XUM, UML, XUMM, XUM.	MOR Editor, which supports the objectization and modeling of requirement engineering.	MOR Editor, which supports the objectization and modeling of requirement engineering.	Extended prototype is required to support software development process.
[3]	UML, ODL, OOA, RE Process Meta-model, XML-based REPM description.	A formal description method of requirements engineering process.	REPM is a unified method for describing different types of processes.	Model does not use RE process Maturity model.
[4]	DFSS, QFD, FMEA.	Requirement engineering process model based on DFSS.	Prioritization through high event interaction coverage	It focuses more on quality improvement than the process improvement.
[5]	Agile development, SOA.	A survey of extreme programming.	Extreme programming approach is best for mission critical system.	The study results cannot be generalized.
[6]	Agile Method, OOP, Extreme Programming.	Highlights developer characteristics for effective agile method.	Developer characteristics for Effective Agile Method.	-
[7]	Extreme Programming.	Highlight properties to tune when designing an extreme programming course.	Best for learning extreme programming technique.	A small release never takes longer than four months.
[8]	Design patterns for computer-mediated interaction. Pair programming.	Social practices for distributed pair programming.	A comprehensive understanding of the impact of plug-ins.	Distributed pair programming has caveat that programmers are not co-located.

[9]	Columbia Video Network, Extreme programming, EJB, CORBA, COM.	Distance learning approach to teaching extreme programming.	Useful for educational purposes.	Virtual presence is the key limitation as participants are not co-located physically.
[10]	Agile Methods, Object-Oriented Metrics.	Agile software development practices.	Suitable for small size projects.	Cost and time factor is not studied.
[11]	Requirement Engineering.	A novel requirement engineering process.	An effective requirement engineering process model for software development.	-
[12]	Mock-up driven fast-prototyping methodology.	Combination of RE techniques to carry out for real life complex project.	Best for complex large-scale software development projects.	Costly for small software projects.
[13]	Dynamic time scheduling, Extreme Programming, Agile method, ERP.	Extreme programming practices and time scheduling interface.	Suitable for small and medium enterprises ERP projects.	-
[14]	Software Capability Maturity Model, CMMI.	RE process improvement model using CMMI.	Smooth transition for practitioners familiar with CMM techniques.	Validation is not applied on R-CMMi Model.
[15]	GORE, SORE, Aspect Oriented RE.	Membrane computing model for SRE activities.	Resultant model determines the number of functional and non functional entities.	-

IV. Conclusion and Future Work

In this study, we have made an attempt to enlighten the understanding of extreme programming approaches and techniques followed by analyzing it with other requirement engineering models. This study may be helpful for software engineers to comprehend the utility of extreme programming approach and understanding the usefulness of requirement engineering. The effectiveness of the requirement engineering process model based on extreme programming is also highlighted in this study. An extreme programming based requirement engineering model that caters for both the pair programming and distributed pair programming along with the performance evaluation metrics of the models is envisaged to be the prospective future direction.

References

[1] Chih-Wei Lu, William C. Chu and Chih-Hung Chang, "A Model-based Object-oriented Approach to Requirement Engineering (MORE)", 31st Annual International Computer Software and Applications Conference (COMPSAC). IEEE. (2007).

[2] Chih-Wei Lu and Chih-Hung Chang, "A Requirement Tool to Support Model-based Requirement Engineering", International Computer Software and Applications Conference. IEEE. (2007).

[3] Jiang and Xuping, "Modeling and Application of Requirements Engineering Process Meta-model", IEEE. (2008).

[4] Megan Florent Tchidi and Zhen He, "The Requirements Engineering Process Model Based on Design for Six Sigma", IEEE. (2010).

[5] Ann Fruhling and Patrick McDonald, "Case Study: Introducing eXtreme Programming in a US Government System Development Project", Proceedings of the 41st Hawaii International Conference on System Sciences (2008).

[6] Kieran Conboy, "Method and Developer Characteristics for Effective Agile Method Tailoring: A Study of XP Expert Opinion". Transaction of Software Engineering Methodology, 20, 1, Article 2, ACM. (2010).

[7] Kai Stapel and Daniel Lubke, "Best Practices in eXtreme Programming Course Design". ACM. (2008).

[8] Till Schummer and Stephan Lukosch, "Supporting the Social Practices of Distributed Pair Programming", CRIWG, LNCS 5411, pp. 83–98, Springer-Verlag Berlin Heidelberg. (2008).

[9] Murphy, "A Distance Learning Approach to Teaching eXtreme Programming". ACM. (2008).

[10] Danilo Sato, Alfredo Goldman, and Fabio Kon, "Tracking the Evolution of Object-Oriented Quality Metrics on Agile Projects". Springer. (2007).

[11] Dhirendra Pandey, U. Suman and A. K. Ramani, "An Effective Requirement Engineering Process Model for Software Development and Requirements Management". DOI:10.1109/ARTCom.2010.24. IEEE. (2010).

[12] Deepti Mishra and Alok Mishra, "Successful Requirement Elicitation by Combining Requirement Engineering Techniques". IEEE. (2008).

[13] Zeng Min and Wang Cheng, "Practices of Extreme Programming for ERP Based On Two-dimensional Dynamic Time Scheduling Interface Method". IEEE. (2009).

[14] Badariah Solemon and Shamsul Sahibuddin, "Re-defining the Requirements Engineering Process Improvement Model". IEEE. (2009).

[15] Velayutham Pavanasam and Chandrasekaran Subramaniam, "Membrane Computing Model for Software Requirement Engineering", 2nd International Conference on Computer and Network Technology. IEEE. (2010).

AUTHORS

Muhammad Khalid is presently pursuing his MS in Computing from SZABIST, Islamabad. He has been serving in the field of Information Technology for the last five years. His research interests include software engineering and requirement engineering.

Sami Ulhaq is a student of Masters in Software Engineering at SZABIST, Islamabad, Pakistan. His focused research areas are software requirement engineering, software quality engineering and global software development.

Muhammad Naeem Ahmed Khan obtained degree in Computer System Engineering from the University of Sussex, UK. His research interests are in the fields of software engineering, cyber administration, digital forensic analysis and machine learning techniques.

