International Journal for Research in Engineering Application & Management (IJREAM)
ISSN : 2454-9150 Vol-04, Issue-09, Dec 2018

A Survey on Agile Software Development Model: Extreme Programming

V S Banupriya,

Assistant Professor, Department of Computer Applications, Chevalier T. Thomas Elizabeth College for Women, Chennai, India. vsbanu@gmail.com

Abstract: Agile software development methodologies are called as the light weight development methods because of the informal, adaptive and flexible approach. An agile process provides numerous benefits include quicker return on investment, better software quality, and higher customer satisfaction. Extreme programming (XP) one of the commonly used agile model that aim to increase a software organization's responsiveness while decreasing development overhead. This research work provides broad overview of the agile model Extreme Programming.

***Keywords* — Agile software development, Agile model, Extreme Programming, Lightweight development, Software Quality.**

I. INTRODUCTION

Agile software development methodologies provide a more efficient and lighter way of development that builds software iteratively and incrementally. Agile models emphasis on changing requirements, customer satisfaction, and team collaboration [1]. Agile software development method is people-focused communication-oriented, flexible, speedy, lean, responsive, and learning [2]. Agile models are collections of best practices and principles of software engineering. These principles are used with different approach that makes them more flexible and adaptive during development. Agile software development models shifted the development focus from process to people and valued things that were neglected in traditional models [3].

The agile software development method includes, Extreme Programming (XP), Scrum, Test Driven Development (TDD), Dynamic System Development Model (DSDM), Feature Driven Development (FDD) and Crystal methods etc. All these agile models follow agile values and principles with some key practices.

A number of agile software development models exist but extreme programming (XP) is one of the most widely used agile model [4]. XP was developed by Kent Beck in 2000 when software industry was seeking for new software development methods to reduce the risk of failure caused by traditional development models. It first stated as "simply an opportunity to get the job done" [5]. After a number of successful trails in practice, the XP methodology was "theorized" on the key principles and practice used.

XP is a test-driven, "light weight" methodology designed for small teams that emphasizes customer satisfaction and promotes team work. XP was created to handle uncertainties in development environment. XP practices are set up to mitigate project risk and increase likely hood of success. The XP can be used for rapid application development of web applications [6].

XP is said to improve the overall product stability and maintainability [7]. It is believed to enable effective software development by allowing organizations to deliver and change requirements quickly during the software engineering process. Advantages of XP over conventional practices include lower management overhead, higher team productivity, happier customers, and shorter release cycles [8].

The remaining part of this paper contains Section-II discusses about existing software development methodologies, Section III explains about life cycle of Extreme Programming. Section IV deals with XP practices, Section V deals with XP values, Section VI deals with limitation of XP and Section-VII deals with the conclusion.

. II. EXISTING SOFTWARE DEVELOPMENT METHODOLOGIES

Over a several decade software development teams used the traditional software development methodologies. As conventional software systems become big and complex software development teams often struggle to produce software that is on time, within budget and with all promised functionalities so a number of development lifecycle models have been created to manage the process.

Waterfall model is the most commonly used process model, in which the various phases of requirements specification, design, implementation, verification, and maintenance are executed sequentially. The Waterfall model has some limitations that the requirements are stable and known at the beginning of the project. As requirements change are

 DOI : 10.18231/2454-9150.2018.1194

inevitable, with frequently changing requirements the approach results to inflexibility [10].

Spiral model overcomes the limitation of the waterfall model [11]. Spiral model have four phases: Planning, Evaluation, Risk Analysis, and Engineering. These four phases are iteratively followed in sequence. However, the spiral model has the limitation that, highly skilled people are required and process is more time consuming and expensive [11] [12].

Agile approach, XP effectively deals with changing requirements, which is difficult to manage in waterfall model. In Agile approach development activities are carried out in small phases, based on collaboration, adaptive planning, early delivery, continuous improvement, regular customer feedback, frequent redesign resulting in development of software increments being delivered in successive iterations in response to the ever-changing customer requirements [13].

III. LIFE CYCLE OF XP PROCESS

The life cycle of XP consists of six phases: Exploration, Planning, Iterations to Release, Productionizing, Maintenance and Death phase (Figure:1)[5].

Exploration Phase: Exploration phase is the first phase of XP life cycle which deals with requirement and architecture modeling of the system. In this phase, user requirements, architecture, tools and technology are defined. A meeting among customer, users and developers is arranged to plan release. Customers write out the story cards that they wish to be included in the first release. Each story card describes a feature to be added into the program. These user story cards comprises of short name, priority of story and one or two text paragraph without technical detail [5]. User story should be detailed enough that help the developers to understand system requirement and also in making estimates.

The exploration phase takes between a few weeks to a few months, depending largely on how familiar the technology is to the programmers.

Planning Phase: Sets the priority order for the stories and an agreement of the contents of the first small release is made. The programmers first estimate how much effort each story requires and the schedule is the agreed upon. The time span of the schedule of the first release does not normally exceed two months. The planning phase itself takes a couple of days. During planning phase decision about team size, code ownership, schedule, working hours are taken.

Iterations to Release Phase: This phase includes several iterations of the systems before the first release. The schedule set in the planning stage is broken down to a number of iterations that will each take one to four weeks to implement. The first iteration creates a system with the architecture of the whole system. This is achieved by selecting the stories that will enforce building the structure for the whole system. The customer decides the stories to be selected for each iteration. The functional tests created by the customer are run at the end of every iteration. At the end of the last iteration the system is ready for production.

Productionizing Phase: This phase requires extra testing and checking of the performance of the system before the system can be released to the customer. At this phase, new changes may still be found and the decision has to be made if they are included in the current release. During this phase, the iterations may need to be quickened from three weeks to one week. The postponed ideas and suggestions are documented for later implementation during, e.g., the maintenance phase.

Maintenance Phase: In this phase new functionality is built while keeping the old one running [5]. New architectural design and technologies can be introduced however XP team has to do more care as the system is in production also. The changes that cause production problems are stopped immediately. The maintenance phase may require incorporating new people into the team and changing the team structure.

Death Phase: This is the last phase of XP. There are two possible situations in which a software system reaches to death phase. In first case, if the developed software has all the needed functionality and customer is satisfied and has no more stories, then it is time to finally release the system. A small document of five to ten pages is created, about the system for future use. In other case, customer may require a set of features that cannot be developed economically. In such situation, it will be better to close the software development which is called entropic death of system [5].

IV. XP PRACTICES

There are twelve XP practices that distinguish XP from other software process models. These practices are used during software development under the guidance values and principles of XP[5].

Planning Game: System requirements are collected on story cards that are used for further planning. Different team roles, team size, working hours and overall schedule is defined during planning game. Planning game is performed in two parts called release planning and iteration planning.

 DOI : 10.18231/2454-9150.2018.1194

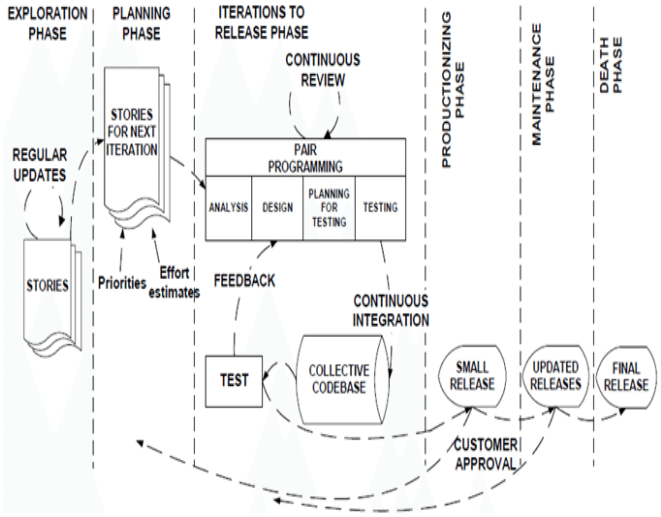

Figure1 Life Cycle of XP Process

Small Releases: In each release a set of requirements are developed that have some business and development value [3]. Small releases make the system open and available for evaluation by the customer. Small releases help in getting immediate customer's feedback about system. Metaphor: It is the architectural design of the system that describes how system should works. For developers, It is very important way to understand the system.

Simple Design: Simple design is a great practice of XP that helps to design basic required functionality of the system and avoids unnecessary details. It focuses on currently needed features not on future requirements. Continuous Testing: Continuous testing provides quick feedback. XP uses unit testing and acceptance testing continuously.

Refactoring: Refactoring is restructuring the system without changing its behavior. It is performed to improve the quality and flexibility of design. It is a routine activity of XP developers to make the code quality better.

Pair Programming: It is very interesting feature of XP that distinguish it from other development approaches. In XP, coding is performed by the two programmers at same machine. The idea behind pair programming is to develop high quality software at lower cost. As most of the errors are captured and corrected within seconds by the companion programmer.

Collective Ownership: Any programmer can access any part of code any time to improve it. This is called collective ownership of code. Code review by number of programmers; enhance the quality of software to be developed.

Continuous Integration: After completing every task, system is integrated and tested. It may happen many times a day. This reduces integration problems and improves software quality.

40-Hour Week: XP discourages extra-long working hours for developers. Tired and bored programmers make more mistakes that's why unnecessary overtimes are avoided in XP. It is a rule of XP, to work 40 hours a week not more than this.

On-Site Customer: A customer's representative is a part of XP team and remains on site all the time. He/ she is usually a domain expert that can decide about system's desired features, answer the questions and can steer the development process. On-site presence help to reduce communication gap between developers and customer. A quick feedback remains available to developers about desired software.

Coding Standards: Coding standards are followed in XP. Code is owned collectively and can be accessed or changed by any programmer. To share the code among programmers, it is necessary to follow some common coding standards.

V. XP VALUES

Extreme Programming (XP) is based on values. Start with five XP's values listed [8] *Simplicity:* We will do what is needed and asked for, but no more. This will maximize the value created for the investment made to date. We will take small simple steps to our goal and mitigate failures as they happen. We will create something we are proud of and maintain it long term for reasonable costs.

Communication: Everyone is part of the team and we communicate face to face daily. We will work together on everything from requirements to code. We will create the best solution to our problem that we can together. *Feedback:* We will take every iteration commitment seriously by delivering working software. We demonstrate our software early and often then listen carefully and make any changes needed. We will talk about the project and adapt our process to it, not the other way around.

Respect: Everyone gives and feels the respect they deserve as a valued team member. Everyone contributes value even if it's simply enthusiasm. Developers respect the expertise of the customers and vice versa. Management respects our right to accept responsibility and receive authority over our own work.

Courage: We will tell the truth about progress and estimates. We don't document excuses for failure because we plan to succeed. We don't fear anything because no one ever works alone. We will adapt to changes whenever they happen.

VI. LIMITATIONS OF XP

Although XP methodology can result in an improved process which is more efficient, predictable more flexible and more fun it also has weaknesses such as [9]

- XP is not suited for difficult and complex projects.
- Pair programming cannot be applied for projects exclusively with one developer.

 DOI : 10.18231/2454-9150.2018.1194

- It needs great amount of coordination amongst the programmers during the course of pair.
- Less focus on design.
- Can result in a never-ending project if not managed properly.

VII. CONCLUSION

The results of the research conducted indicate that the XP approach to software development is far better and productive as compared to traditional software development methods. It has several features and aspects to support projects for large or small organization, and the projects that's need short or long period of time to be finished. This model used best practices in agile fashion to accommodate rapid application development needs. Despite of these advantages there are some limitations also. XP's 12 core practices are closely related, and implementing only a few will not necessarily bring all potential benefits. Having a full-time on-site customer is sometimes impractical. Overall, XP a agile software process that speeds up development and lets teams react flexibly to requirement changes, but some issues remain.

REFERENCES

[1] L. Williams, "Agile software development methodologies and practices," in Advances in Computers, vol. 80, Elsevier Inc. 2010, pp.1-44.

[2] Manish Kumar, (2015)" A Detail Study of Agile Software Development with Extreme Programming, International Journal of Advanced Research in Computer Science and Software Engineering

[3] D. Cohen, M. Lindvall, and P. Costa, (2004) "An introduction to agile methods." ADVANCES IN COMPUTERS, vol. 62, 62, pp.166

[4] M. R. J. Qureshi and J. S. Ikram, "Proposal of Enhanced Extreme Programming Model," International Journal of Information Engineering and Electronic Business, vol. 7, no. 1, p.37- 42, 2015.

[5] K. Beck, Extreme Programming Explained: Embrace Change, Addison-Wesley, Boston, MA, USA, 1999.

[6] Maurer, F, Martel S , (2002) "Extreme Programming Rapid Development for Web Based Applications",

[7] Poole, C.; Murphy, T.; Huisman, J.; and Higgins, A. Extreme maintenance. In G. Canfora and A. Amschler Andrews (eds.), Proceedings of the Seventeenth IEEE International Conference on Software Maintenance. Los Alamitos, CA: IEEE Computer Society Press, 2001, pp. 301–312.

[8]. Cao, L.; Mohan, K.; Xu, P.; and Ramesh, B. How extreme does programming have to be? Adapting XP practices to large-scale projects. In R.H. Sprague (ed.), Proceedings of the ThirtySeventh Hawaii International Conference on System Sciences. Los Alamitos: IEEE Computer Society Press, 2004.

[9] John Noll, Darren C. Atkinson, Comparing Extreme Programming to Traditional Development for Student Projects: A Case Study, Department of Computer Engineering Santa Clara University, year unknow.

[10] Pressman R.S., „Software Engineering: A Practitioner"s Perspective", 5th ed., McGraw- Hill, New York, 2000, pp. 769-798.

[11] Boehm B, „A Spiral Model of Software Development and Enhancement",(1986) ACM SIGSOFT Software Engineering Notes, ACM, 11(4):14-24

[12] Jalote, Pankaj, Aveejeet Palit, Priya Kurien, and V. T. Peethamber, (2003) "A Process Model for Iterative Software Development." Infosys Technologies Limited Electronics City, Bangalore-561 229

[13] Matharu, Gurpreet Singh, Anju Mishra, Harmeet Singh, and Priyanka Upadhyay, (2015) "Empirical Study of Agile Software Development Methodologies: A Comparative Analysis." ACM SIGSOFT Software Engineering Notes 40, no.1

 DOI : 10.18231/2454-9150.2018.1194