Volume 5, Issue 1
February 2016

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X

Software Development Lifecycle for Extreme Programming

Aroosa Hameed

Department of Computer Science, University Of Gujrat, Sub Campus Satellite Town, Rawalpindi

E-mail:aroosahameed379@yahoo.com

ABSTRACT

The software engineering offers a chance to the countries to create dramatic improvements in economic development. The Pakistan software business also plays a significant role in strengthening the economy. Extreme Programming (XP) is an agile methodology providing quality products and provides a chance to retort to ever-changing client needs. Every software methodology has its own development cycle consisting of the testing part at the tip of the event. To find an answer of ever dynamical requirements, developing a framework which will consists testing at each phase to realize the international standards. Also, providing an accurate roadmap of the framework. Careful review of XP literature area unit typically done to spotlight problems like requirement changes at the tip of development. Existing XP models are studied to determine their strengths and weaknesses and ultimately projected a new XP framework with testing as fundamental part is conferred to resolve addressed problems. The proposed framework makes XP *be* useful for small teams of specialists, who are able to communicate well with the end-users and who are smart designers and implementers thus providing productivity and maintenance practices for developing quality products which could facilitate to comprehend the international standards.

Keywords: *Software Development Life Cycle (SDLC), XP, Agile methodologies*

1. INTRODUCTION

Software Development Life Cycle (SDLC) is a strategy of constructing or maintaining software systems [1]. Commonly, SDLC incorporates phases from investigation to developed code, testing and maintenance. Different methodologies are utilized by development groups to build up the products and these methodologies shapes the framework for the whole development method. As of now, there are two SDLC strategies which are utilized by most system developers, specifically the traditional development and agile development [2].

Software methodologies like Waterfall, Vee-Methodology and Rational Unified Process (RUP) are called traditional software development methodologies and these are classified as heavyweight methodologies [3]. These

Methodologies supported a sequential series of development steps. There are four phases which are characteristic of the traditional software development approach. The primary stage is to assemble the prerequisites from clients toward the start of the undertaking. Once the prerequisites are taken off, the future step is the design and architectural planning where a technical infrastructure is shown in the form of diagrams. Once the team is satisfied with the design plan, the next phase of the project is the development phase where code is composed. The testing phase may overlap with the development phase to guarantee issues can be determined before.

The traditional software development methods are dependent on a set of predetermined processes. The achievement of a project which is approached in this manner relies on knowing the greater part of the requirements before the development of the project begins, means that if any client need is altered throughout the development lifecycle will generate the issues.

Agile development depends on the idea of incremental and iterative development, in which stages within a development life cycle are checked on. In this methodology software is enhanced iteratively by using client input to converge on solutions [4]. In agile methodologies, the life cycle of software development is divided into smaller parts, called “increments” or “iterations”, in which each of these increments work on each of the stages of development. The major factors of agile are: Initial customer involvement in the project, Iterative approach of development, Self-organizing groups, Adaptation to change. There are as of now six techniques that are recognized as agile development strategies, which are: Agile Crystal methodologies, agile dynamic software development methodology, feature-driven development method, lean software development, scrum, extreme programming.[5]

Extreme Programming (XP) is one of numerous new lightweight software development life cycle methodologies. XP is used to create intricate large scale software of satisfactory quality in a reasonable measure of time. XP is characterized by values, activities, and practices. XP gives a very close coordination between the programmers and the clients.

Volume 5, Issue 1
February 2016

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X

The client defines the significance of the software that is to be created and provides the prerequisites as client stories. A user story is a non-technical representation of how the user will utilize the system/program to fulfill the business needs.

The software engineer reacts with an estimate of the time to deliver a prototype that may meet the minimum requirements. At initial, the estimate will be very rough yet with successive iterations, it becomes dependable. In the iterations, the client gives input that how much the software has addressed their needs and what additional user stories are required for the next iteration. The developer also keeps the client educated about the technical risks of executing every story. After deployment, XP can still give value by utilizing refactoring means enhancing code without changing the system output to enhance efficiency and have built-in testing to guarantee the code quality under the maintenance.

Our research questions are:

1. If requirements change even late in development, can this XP process Welcome?
2. Does the quality of project increases when a group utilizes XP Practices and values?
3. Is Test Driven Development can have a positive effect in XP Process?
4. Because of Test Driven Development is a there decrease in defect rate?

2. RELATED WORK

The literature on Extreme Programming is fundamentally centered around audit of the practices, and how this is unique in relation to traditional methodologies. In response to traditional approaches, new lightweight methodologies showed up. Lucas Layman, Laurie Williams, Lynn Cunningham gave a review of Extreme Programming [6]. In their research paper, they depicted Extreme Programming (XP) is the best known of the lightweight strategies. XP works best when applied to small, co-located team's under ten individuals. Various reports examine the utilization of XP with small teams. Wood and Kleb [7] formed a two person XP team and investigated the profitability of their venture as a component part of a study at NASA. At the point when the project results were compared with past projects, the XP approach was around twice as profitable.

XP has four key values: communication, feedback, simplicity, and courage. A related research paper by the Gerald DeHondt II, Alan Brandyberry [8] portrayed how these values actualize the best practices of past Systems Development Methodologies. The twelve XP practices [9] are: planning, small releases, metaphor, simple design, refactoring, testing, pair programming, collective ownership, continuous integration, 40-hour week, on-site customer, and coding standards. Robinson and Sharp [10] performed a participant observer study. The researchers took an interest with an XP team to look at the relationship between the 12 XP practices and the four XP values. This research paper inferred that the XP practices can be utilized to make a community that supports a culture that incorporates the XP values. XP concentrate on the individual as the essential drivers of development success. They are those closest to the solution and ought to be knowledgeable about how the solution will actualize. McKeen, Guimaraes, and Wetherbe [11] contend that client cooperation, enhance the quality of the system in several ways, for example, giving a more exact and finish prerequisites, maintaining a strategic distance from the development of unimportant features, and enhancing client comprehension of the framework. Kent Beck [12] states that the programming strategy of XP is to keep the code simple to alter. Iterative development requires that each developer must release code at any rate, once per day after passing all the unit tests or completing a smaller part of planned functionality [13]. This recognizes issues early and guarantees everyone is working with the most recent version of the system.

One remarkable distinction in the middle of XP and other methodologies is its emphasis on rule of testing. Testing is the premise of all development. In fact, XP programmers are required to compose tests as they compose production code. Marick [9] has recommended another model for Test Development. In addition to the documentation; testers use different wellsprings of information while designing tests. The tester is in charge of taking manageable action in response to changed documents or changed codes. One example of the benefits of the XP methodology is the Chrysler Comprehensive Compensation System (C3) (Highsmith, 2000, February). The venture was begun in 1990s and was being developed in Smalltalk. In 1996, the undertaking was stuck in an unfortunate situation because of a low quality code. At that point, the code was discarded and the undertaking was restarted utilizing XP as its methodology. Taking after this rerouting, the principal phase of C3 went live in mid-1997. At present, the object oriented (OO) payroll system comprises of 2,000 classes and 30,000 approaches. At last, XP is intended to permit small development teams to deliver rapid, change rapidly, and change regularly. XP provides the set of practices that empower small development teams to work successfully in today's environment of rapid development. Further work in this area is ongoing. Extreme Programming is not the answer for all issues; it additionally has its disadvantages and downsides. Therefore, Extreme Programming cannot be applied successfully in each sort of programming venture.

Volume 5, Issue 1
February 2016

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X

3. PROPOSED MODEL

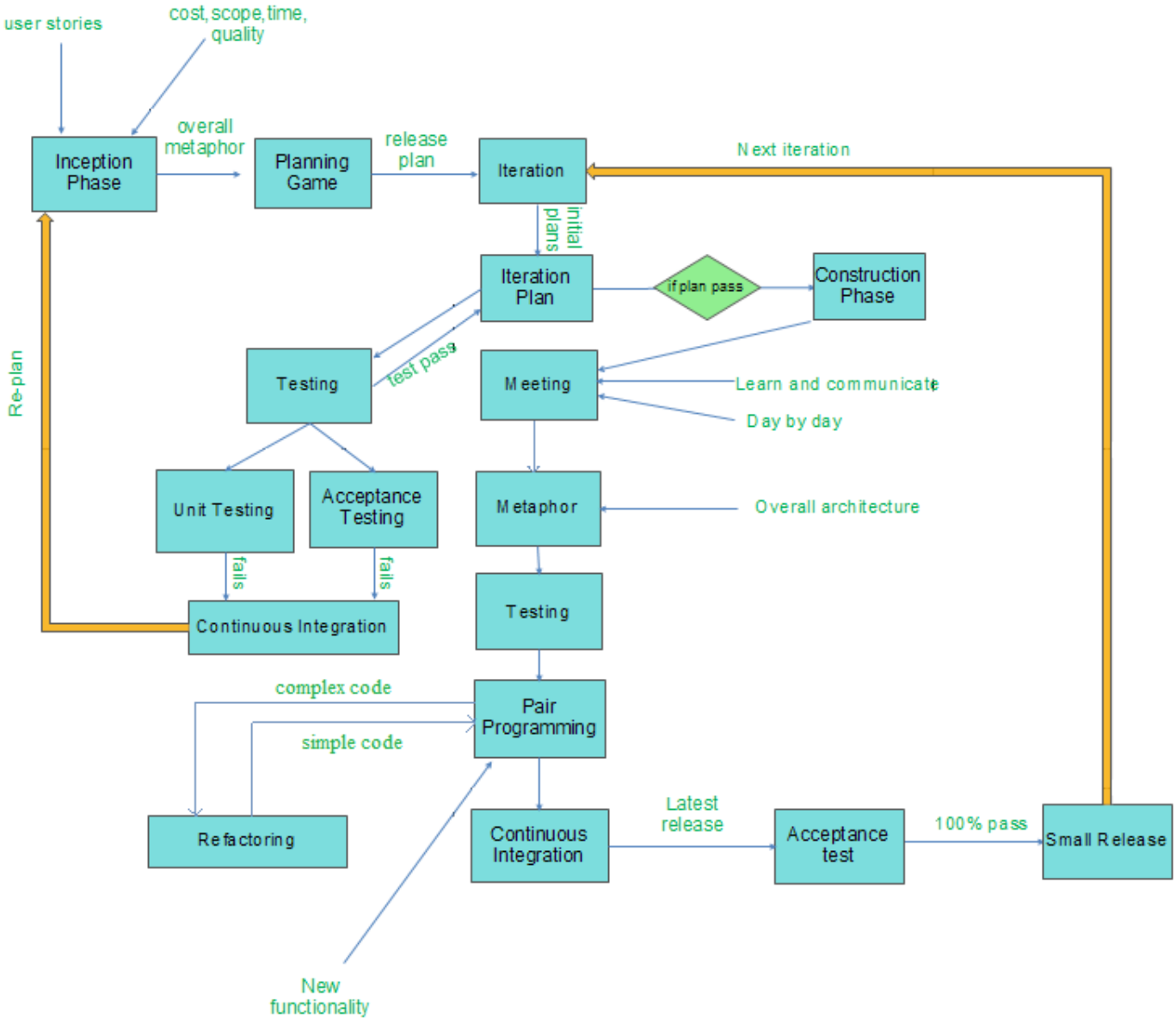

Figure 1: Proposed Model

Volume 5, Issue 1
February 2016

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X

DESCRIPTION OF PROPOSED HYBRID MODEL:

Extreme Programming is an **iterative** and **incremental** procedure implies that it gives small incremental releases. The overall project is partitioned into smaller parts that delivers an increment in functionality and called "Small Releases". A **small release** is a more recent version of planned framework that provides some new functionality. All functions incorporated into the small releases are satisfied completely. An XP project creates **latest releases** every one to three months to gain feedback early. Therefore the system grows over time with releases. Releases are negotiated in the Planning Games. The on-site customer characterizes what ought to be part of the most recent release and the developers determines amount of time it will take to actualize the release. Each release cycle comprises of a couple of iterations, each of which is at most three weeks in length. The iteration is an organizational utility used to facilitate the planning.

We isolate the process in four stages:

Phase I

Inception Phase:

During the inception phase, you should set up:

- Estimates of 4 variables, i.e.: Cost, Scope, Time, and Quality.
- A general perspective of the project's prerequisites (user stories), key features and preparatory project synopsis (metaphor).
- An introductory risk evaluation.

Estimates:

The variables which XP identifies for software development project are:

1. Cost

How much amount of money is to be spent. The assets, including developers, equipment's, and so forth accessible for the project are openly identified by this variable.

2. Time

This variable decides when the most recent small release ought to be given.

3. Quality

The correctness of the system means how much functionality performed by system as defined by the customer and how well tested it will be.

4. Scope

Depicts what and how much will be done (functionality).

User Stories:

User stories are somehow same as use cases. They are used for assessing time for the release plans. User Stories are composed by end customers, including features that the framework must incorporate. They are about sentences of content composed by the client as indicated by to client understandings. User stories just give a small viewpoint so developers can only gauge time to actualize the story. For further details developers will go to the customer and receive requirements face to face. Each story will get an evaluation in this manner, providing development time of the story. This development time is amount of time is expected to actualize the story in code. Stories is a focus on user needs.

One of the major requirements of XP is to have an on-site customer which is a part of the development team. All stages of an XP project require communication with the end customers. User stories are composed by the customer, with developers that helps to make assessments of above depicting four variables.

Overall Metaphor

Gives an introductory entire framework design by understanding user stories.

Phase II

Planning Game:

Develop a plan by taking planning as a game (as the name proposes). The planning game has a goal, playing pieces, players, and playing rules for reasonable moves.

- **Goal:** The goal of the game is to put user stories of the highest priorities into production over the full life of the game.
- **Pieces:** The essential piece of the game is the user story. Developers can easily develop plans from lists of user stories and make gauges.
- **Players:** The players are on site customers and Developers of the project.
- **Moves:** This incorporates following**:**
  - ✓ **Write Story**: For the purpose of playing game, customers can compose stories at any time that should portray some functionalities.
  - ✓ **Estimate Story:** Development team of venture takes every story and assigns its priorities. If the estimate becomes higher, then the story is partitioned into smaller stories and after that work begins on it. In the event that the assessment is lower, combines it with another story.
  - ✓ **Make a Commitment:** On site customers and Developers cooperate to choose which stories have highest priority and constituted the next release and estimates when it will be prepared to put into production.

The result of the planning phase is:

A development plan for the overall project, demonstrating iterations and evaluation criteria for each iteration.

Release Plan:

The plan of the entire framework can be released to developers as well as customers of the framework. Before developers can begin taking a shot at it the on-site customers can survey it at least once to figure out if a given plan can perform desired functionality or not.

Phase III

Iteration:

Any change required in release plan should be possible in iteration phase. It incorporates 3 steps:

1. Iteration plan:

In the event that an on-site customer needs any adjustment in the plan, then an "iteration plan meeting" is held to regenerate the plan of programming tasks. User stories of high priorities are chosen from the previous release plan. Tasks that are duplicated can be evacuated. These concluded tasks will be the detailed plan for the iteration. Then a plan can be given to Developers to estimate time to finish tasks and afterward, perform the tasks. So in this way the new iteration plan is

Volume 5, Issue 1
February 2016

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X

made which is also called initial plan. This Iteration plan goes for further testing to customers and developers.

2. Testing:

- **Acceptance Testing:**

Acceptance tests are made from the depictions or stories given by the client. In this phase acceptance tests are created for iteration plans testing. The client makes acceptance testing, when plans are prepared by the given user stories. Iteration plans must tested by acceptance testing to guarantee whether to move to the next stage or not. One or more acceptance tests are made by clients for guaranteeing that the correct functionalities are incorporated into the iteration plan and how they function as per plan.
If the plan passed by the acceptance test given by users, then it actually shows that the functionalities incorporated in the plan must be actualized in small release. Customers are actual judges because they judge whether plans are as per their sought needs or not and reviewing test scores to choose which plans are passed and which are not and after that developers prioritizes each functionalities given in the plans. Until plans do not pass the acceptance tests, these are not executed by developers. This means that new acceptance tests must be created for each of the iteration plan.

Acceptance tests ought to be passed so that plans are implemented and this is one of the biggest challenges for planners of the project. Quality Assurance of small release actually relies upon testing phase. It is the team's responsibility to oversee time variable if the iterative plan cannot pass acceptance tests and then continuous integration can be done by both planners and on-site customers.

The acceptance tests are used for checking whether plan includes the functionalities provided by user stories or not. The passed acceptance test reflects that a customer's requirements have been met and the iterative plan is acceptable so move on the next phase.

- **Unit Testing:**

Unit tests are actually testing for coding that is developed by the programmers, however, in this phase these tests are used for checking whether plans that are created have proper workflow or not. The criteria for unit test of iteration plan is actually set by taking into consideration of SDLC phases. If unit testing is failed, then revamped the workflow for the project. Ensure that this new workflow did not introduce any change in the functionalities, but if unit testing is passed then you have a proper workflow for project development, thus saving time variable which is a most challenging factor. If you want that your iteration plan pass out all of unit tests, then ensures that a proper workflow is provided for all functionalities included in the plan.
If unit testing fails, then your latest release plan is incompatible and you have to rebuild plans. Unit testing is done after the acceptance test has passed as acceptance test, demonstrate functions included in plan, unit tests demonstrate a workflow in which these functionalities can be actualized by developers of the project.

3. Continuous Integration:

We have about 3 cases.

Case I:

If acceptance test is passed and unit test failed, then planners rebuild plan with same functionalities but different workflow.

Case II:

If an acceptance test fails, then again rebuild plan as no unit test done in this case.

Case III:

If the iteration plan failed both types of testing, i.e. acceptance testing and also unit testing, then continuous integration can be done. Continuous integration at this phase can be done by both on-site customers and planners as well. On-site customers present their refined user stories at one repositories and planners likewise make a workflow for each user story that how each functionality is planned to be executed and put at same repositories. Then integration can be done. Planners ought to be integrating iteration plans, workflow into the repository every few hours in the meantime, customers check out the plans and performed the acceptance testing. Continuous integration regularly maintains a strategic distance from development of plan in fragments, where planners and on-site customers are not communicating with each other about what can be re-used, or what functionalities could be included in which type of workflow. In continuous integration step every on-site customer needs to perform acceptance tests on recent variant and planners likewise perform unit tests on the most recent version.

If only a small portion of the iteration plan with the desired functionality passed testing, then re-plan other portions thus integrating other tested portions with that portion of iteration plan. Always work with the latest version of the iterative plan so you may discard all past day versions every day.

Phase IV

Construction Phase:

During the construction phase, Iteration plan is implemented and put into a small release. Construction Phase is actually manufacturing phase of the undertaking. In construction phase team needs to create code alongside overseeing resources and controlling estimated variable costs, scope, time and quality. Construction phase outcome or result is a small release that might give to on-site customers of the project. At minimum, it comprises of:

- The user manuals for users.
- An explanation of the current release.

After getting the outcome team must see whether customers satisfies without putting the project to higher risks. This initial release is often called a "beta" release. Then testing phase can be done. After 100% testing is done, then correct release is provided. Construction Phase consists of following steps:

1. Meeting:

All project members communicate effectively and imparted workflow plan with each other in the form of meetings. A meeting should be possible once in a week or twice a week, however standup meetings are hung consistently. Every morning of working day there is a standup meeting held to share problems occurred in the workflow of the plan and resolve problems avoiding long discussions. A standup meeting requires everyone to attend instead of only a few developers. Standup meeting demonstrates who actually

Volume 5, Issue 1
February 2016

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X

contributes and who do not. Motivation behind the stand-up meeting is that the developers replied at least three questions:

- Which tasks were done yesterday?
- What will be today tasks, goals and work arrangements?
- What issues are bringing on deferrals and dangers?

2. System Metaphor

System metaphor is overall architecture or design of the release. After a standup meeting held, overall tasks of a working day are talked about and then a metaphor is outlined. If new members are included in team of the project, then just a system metaphor about overall project can disclose to them, thus you don't have to explain him or giving him a colossal measure of reports. So that new people, begin contributing quickly. Another motivation behind a system metaphor is utilization of names for classes and objects which helps you in code reusability. On the off chance that name of some item or class as of now exists you do not use it again, thus saving the time. Make the design or metaphor easier to comprehend, most straightforward approaches may be followed so everyone understands easily.

3. Testing:

Testing is most important part of this XP Framework. After a system metaphor is designed it must be tested to guarantee that the project moves in the direction the client needs it to be. At this testing level, it is to be test that which objects and classes are defined and used, which diagrams can be used as a design and tests the relationships between objects and classes
Testing can be executed as a code or using a proper testing framework. Here testing is of architectural plan of small release so no customers are involved in it.
On the off chance that metaphor is satisfactory and taking the project in the right direction and also actualizing all correct functionality, then moves to the next step of the construction phase. In the event that it is not at satisfactory level then changes can be made or reconstruct the system metaphor.

4. Pair Programming:

Two or more individuals work out together to program a code for the functionalities and tasks to be implemented and this can be done on a solitary PC. Pair programming increases the quality of code and also deliver the code at a time instead of the individuals working independently.
What is the quality of the code?
The best answer is that you have code that perform project functionality with less no. of lines of code and much easier approach is used for this.
Quality code can be achieved by pair programming. Programming is a skill that requires significant investment of time to learn so when two individual deal with the code next to each other in front of a single computer, then both of these put all their efforts to program quality code.
Pair programming likewise incorporates important part called "refactoring".

- **Refactoring:**

Refactoring is actually removing redundancy, eliminating unused functionalities and making code as simple as could be allowed without influencing consequences of the code. Refactoring can be done throughout the pair programming so that it saves time and increases quality of the code. To remove complexity developers use refactoring of code. If one pair of programmers generates the code, then another pair may refactor the code, accordingly providing simplest code which is easy to understand and take lesser time to survey and test.
One other aspect of pair programming at here is that if any new idea that developers needs to actualize can likewise be added to the code as small snippets.
For pair programming to be effective programmers must have potential to say other partner "how about we attempt your idea first." Experienced programmers have such potential to give another programmer chance first. When a project team uses pair programming at first they feel some awkward but after then they become used to.

5. Continuous Integration:

At here continuous integration means doing some changing in the code or integrating coding of paired programmers, thus executing a single task but this did not influence the result. Once coding can be done by pair programming all codes composed by pair programmers can be combined and stored in the codebase. Continuous integration can be done by the developer's team of the project. Continuous integration can be done in every couple of hours. Always used the most recent adaptation of code each time. In fact, it is part of the construction phase, so throughout the construction of code it may occur.
The benefits of continuous integration are:

- If any new change is made, the code becomes easier.
- If problems occurred during testing, continuous integration can cover these problems.

In XP, if changes are made they are made into codebase every couple of hours, but at least once in a day. A developer will only perform continuous integration when he or she:

- Added some new functionalities.
- Refactors the coding part.
- Fixes some errors or bugs.

After continuous integration can be done the next step is a beta release. Beta Release is generated where all code put together.

Beta Release:

Small release before final testing is called "beta release". At that time beta version was made ready for final testing, which is considered the biggest challenge of the project.

Acceptance Testing:

In this phase, final testing of beta release can be done. If beta version passes the acceptance test 100%, then it became a final small release. Acceptance test is done by the customers and developers both at this stage. On-site customers test the desired functionalities and developers test whether the techniques and methods that are used to implement functions work properly.
At this stage, even test can be done by adding some errors in the version and seeing whether these errors triggered or not.

After 100%assurance by both developers and customers, small release is put up.

Small Release:

Smaller version or releases are put up in iterations for the development of the overall project. Small releases may be developed every day or every week depends upon how much complex your system is. Each small release at the end is a demonstration of the user stories provide.

Next Iteration:

When small release is created, then the next iteration occurs. In next iteration next part of the planning game can be performed as another small release. At the end each of these small releases together make up a project that is according to user needs.

4. CONCLUSION

The aim of this research paper was to propose a suitable Extreme Programming framework for resolution of issues like requirements changing. This proposed framework focuses the development of quality software by small groups by using testing at each stage of the development phase. The proposed framework contains features of SDLC and scrum methodology. I believe that the implementation of this framework will help the Pakistani software industry to improve the productivity of a team and to develop quality products. However, I acknowledge, much work remains to further validate and extend this framework.

5. ACKNOWLEDGMENT

I would like to thank all those who dedicated their energies, resources and time to the success of this research. The foremost, thank you goes to my ever encouraging teachers. Last but not the least my thanks goes to my friends and family for their help, support and efforts.

REFERENCES

[1] Systems Development Lifecycle: Objectives and Requirements. Bender RPT Inc. 2003.

[2] http://www.ambysoft.com/essays/agileLifecycle.html

[3] Nikiforova, O., Nikulsins, V., Sukovskis, U.: Integration of MDA Framework into the Model of Traditional Software Development. In: Frontiers in Artificial Intelligence and Applications, Databases and Information Systems V, vol. 187, pp. 229–239. IOS Press, Amsterdam (2009)

[4] Szalvay, Victor. An Introduction to Agile Software Development. Danube Technologies Inc. 2004.

[5] Programming in the eXtreme: Critical Characteristics of Agile Implementations Gerald DeHondt II_, Alan Brandyberry_Management & Information Systems Department, Kent State University gdehondt@kent.edu, abrandyb@kent.edu

[6] Exploring Extreme Programming Lucas Layman, Laurie Williams, Lynn Cunningham North Carolina State University, Department of Computer Science, {lmlayma2,lawilli3}@ncsu.edu Clarke College, lynn.cunningham@clarke.edu

[7] W. Wood and W. Kleb, "Exploring XP for Scientific Research," *IEEE Software*, vol. 20, pp. 30-36, 2003

[8] Programming in the eXtreme: Critical Characteristics of Agile Implementations Gerald DeHondt II, Alan BrandyberryManagement & Information Systems Department, Kent State University e-Informatica Software Engineering Journal, Volume 1, Issue 1, 2007

[9] Extreme Programming Sergey Konovalov and Stefan Misslinger May 23, 2006

[10] H. Robinson and H. Sharp, "XP Culture: Why the twelve practices both are and are not the most significant thing," presented at 1st International Agile Development Conference (ADC '03), Salt Lake City, UT, 2003

[11] J. McKeen, T. Guimaraes, and J.Wetherbe. The Relationship between User Participation and User Satisfaction: An Investigation of Four Contingency Factors. MIS Quarterly, 18(4):427– 451, 1994.

[12] K. Beck and C. Andres. Extreme Programming Explained: Embrace Change 2nd Ed. Addison- Wesley, Boston, 2004.

[13] S. Joosten and S. Purao. A Rigorous Approach for Mapping Workflows to Object-Oriented IS Models. Journal of Database Management, 13(4):1–19, October–December 2002.

[14] B. Boehm and R. Turner, "Using Risk to Balance Agile and Plan-Driven Methods," *IEEE Computer*, vol. 36, no. 6, pp. 57-66, June 2003.

AUTHOR PROFILES

Aroosa Hameed received her BS degree in Information Technology from University of Gujrat, in 2016. She is a research student of University Of Gujrat. She has done his research by introducing best practices in different software development process models.