Volume 5, Issue 10, October-2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Detail Study of Agile Software Developnment with Extreme Programming

[1]Manish Kumar, [2]Santosh Kumar Singh, [3]Dr. R. K. Dwivedi
[1, 2] Assistant Professor, University Department of Computer Applications, Vinoba Bhave University, Hazaribag, Jharkhand, India
[3] Associate Professor, University Department of Mathematics, Vinoba Bhave University, Hazaribag, Jharkhand, India

Abstract- Under the current economic conditions many organizations strive to continue the trend towards adopting agile processes, in order to take advantage of the numerous benefits that these can offer. Those benefits include quicker return on investment, better software quality, and higher customer satisfaction.
While there are numerous studies of agile development in academic and educational settings, there has been Little detailed reporting of the usage, penetration and challenges of extreme programming agile methodologies in traditional, professional software development organizations. This paper explains the differences between traditional software development methods and agile software development methods, and introduces the characteristics of one of the popular agile methods i.e, extreme programming. Along with that we conducted a web-based survey of employees who are directly involved in the production of software from three different companies. The insights presented in the paper can be used in organizations that are in the process of agile software development using Extreme Programming.

Keywords- Traditional software development, Agile software development, Scrum, Extreme programming(XP), Crystal, Dynamic Software Development Method (DSDM).

I. INTRODUCTION

Agile software development (ASD) methodologies [1] have been gaining acceptance among mainstream software developers since the late 1990s, when they were first postulated in the forms of Scrum [2], Crystal [3], Extreme Programming [4] and other methodologies. Today they are established to varying degrees in the academic, educational and professional software development communities. Agile software development methods have been developed and evolved since early 1990s. Due to the short development life cycle through an iterative and incremental process, the agile methods have been used widely in business sectors where requirements are relatively unstable.

Agile method is a software development method that is people-focused commications-oriented, flexible (ready to adapt to expected change at any time), speedy (encourage rapid and iterative development of the product in small releases), lean (focuses on shortening timeframe and cost and on improved quality), responsive (reacts appropriately to expected and unexpected changes), and learning (focuses on improvement during and after product development)

Extreme programming (XP) is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements.As a type of agile software development it advocates frequent "releases" in short development cycles, which is intended to improve productivity and introduce checkpoints at which new customer requirements can be adopted.The methodology takes its name from the idea that the beneficial elements of traditional software engineering practices are taken to "extreme" levels. As an example code reviews are considered a beneficial practice; taken to the extreme, code can be reviewed continuously i.e, the practice of pair programming [5].

In this paper our main aim is to discover in detail about the characteristics of extreme programming methodology and to conduct a research on three IT companies using both the qualitative and quantitative methodologies. The rest of the paper is organized as follows: Section-II deals with the background part, Section-III deals with research design and survey conducted, Section-IV deals with extreme programming life cycle, Section-V deals with extreme programming practices, Section-VI deals with extreme programming values, Section-VII deals with benefits of extreme programming, Section-VIII deals with challenges of extreme programming, and Section-IX deals with the conclusion part.

II. BACKGROUND

A. Agile software development (ASD) methodologies

Agile methodologies include:

1. Scrum: SCRUM methodology was initiated by Ken Swaber in 1995. It was practiced before the announcement of Agile Manifesto. Later, it was included into agile methodology since it has the same underlying concepts and rules of agile development. SCRUM has been used with the objective of simplifying project control through simple processes, easy to update documentation and higher team iteration over exhaustive documentation .It focuses on project management in situations where is difficult to plan ahead, with an importance on feedback mechanisms.[2]

2. Extreme programming: Extreme Programming was introduced by Kent Beck in 2000. XP is a lightweight programming methodology that supports a team by writing software that contains fewer bugs and therefore takes shorter time. This methodology consists of several values, principles and practices to be considered. Some of these must be applied by the whole company, some by the team and most by each and every member of the team. The two essential aspects of extreme Programming are: [4]

(I) short term and very specific programming goals

(ii) an awareness of the social aspects of programming

3. Crystal methodologies: Crystal Methodologies were established by Alistar Cockburn in 2000. They concentrate on efficiency and habitability as components of project safety. Each of the Crystal methodologies requires certain roles, policy standards, products and tools to be adopted. Crystal Clear, which is one of the Crystal methodologies, can be applied to development teams of six to eight members, working on non-life critical systems. It focuses on people, not processes of artifacts.The most agile method, Crystal Clear, focuses on communication in small teams developing software that is not life-critical. [3]

4. Dynamic Software Development Method (DSDM): DSDM (Dynamic Systems Development Method) is a robust Agile project management and delivery framework that delivers the right solution at the right time.The DSDM Philosophy is that any project must be aligned to clearly defined strategic goals and focus upon early delivery of real benefits to the business. DSDM is vendor-independent, covers the entire lifecycle of a project and provides best practice guidance for on time, in budget delivery of projects – with proven scalability to address projects of all sizes and for any business sector. [6]

B. Comparison between traditional approach and agile approach to software development

[7][8][9][10]

Traditional Approaches	**Agile Approaches**
Deliberate and formal, linear ordering of steps, rule-driven.	Emergent, iterative and exploratory, beyond formal rules.
Optimization is the goal.	Adaption, flexibility, responsiveness is the goal.
In this type the environment is taken as stable and predictable.	In this type the environment is taken as turbulent and difficult to predict.
Sequential and synchronous process.	Concurrent and asynchronous process.
It is work centered process because people will change according to different phases.	It is people centered process, as the same team is developing throughout.
Project lifecycle is guided by tasks or activities.	Project lifecycle is guided by product features.
Documentation is substantial.	Documentation is minimal.
Developers do waiting until the architecture is ready.	The whole team is working at the same time on the same iteration. Good coordination between team members
Too slow to provide fixes to user.	Provide quick responds to user feedback
Change requirements is difficult in later stages of the project	Can respond to customer requests and changes easier
More time is spent on design so the product will be more maintainable. The “what ifs” arise earlier	There is no time for the what ifs
No communication within the team, novices stay in their rooms and try to understand things	High level of communication and interaction, reading groups, meetings
Restricted access to architecture	The whole team influences and understands the architecture. Everybody will be able to do a design presentation.
Documents and review meetings are needed to solve an issue	5 minutes discussion may solve the problem
Everything is up front, everything is big before you start	The focus is on whether customer requirements are met in the current iteration

C. Software developnment success rate [11]

This survey was performed in early August 2007 and there was 586 respondents. The survey was announced on the Dr. Dobb's Journal (DDJ) [11] mailing list. Respondents reported that Agile software development projects [1] have a 71.5% success rate, traditional projects [11] a 62.8% success rate, and offshored software development projects [11] a 42.7% success rate. These figures are summarized in Figure 1.On average agile software development strategies appear to be the most effective ways of working. Agile (think Scrum or Extreme programming) approaches are statistically more successful as compared to other software development method. Similarly Traditional (Waterfall) and Data Warehouse(no defined process) strategies are also statistically similar regarding success rates. Offshoring success rate is less as compared to the other three software development method.

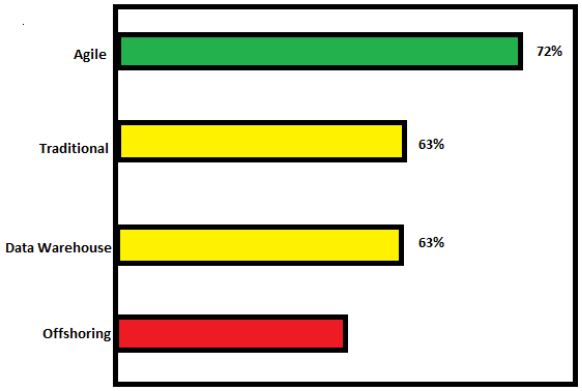

Fig1: Software development success rate

D. Extreme programming methodology

Extreme Programming was introduced by Kent Beck in 2000. Being an emerging agile methodology, XP offers a number of practices, values and principles which are advised to be adopted in order to run a software development project. XP is a package of several practices and ideas, most of which are not new. The combination and packaging of all of these is, however, new. Extreme Programming was in fact targeted especially at small co-located teams developing non-critical products. It has been suggested that the early adopters of agile methods have been small high-tech product companies. Currently, however, it has already been proven at many companies of all different sizes and industries worldwide. XP provides a list of simple, specific, and seemingly naïve principles and values that guide the software development process throughout the main four phases of software development: planning, coding, designing, and testing. The main purpose is to deliver what the customer needs, at the time it is needed. In addition to this, one of the main reasons of its success is its ability to accept changes at anytime during the development. XP also emphasises teamwork; experiences from all stakeholders are employed to meet the specific goals, and within the given constraints. [4]

Extreme Programming is different from traditional approaches in many ways: [4]

1. XP involves short, tight iterations of building and releasing software.
2. Requirements are gathered in terms of 'stories'. These usually begin with 'I want…' or 'We want…'. These are produced by the customer or client. The developers look at the stories and estimate how long they think it will take to satisfy them. Developers and customers negotiate what requirements can be delivered in the time available. This exposes the customer to developers' estimates, and encourages them to prioritise their requirements since it is not possible to do everything in one iteration. The activity of estimating and prioritising to see what can be done in the next iteration is called the planning game.
3. The customer is on site and is part of the development team. Once the iteration has started , developers work in pairs. No development is done by a single individual. All development is jointly owned. This is a surprisingly element of XP.
4. A 'test-first' philosophy prevails. In XP you write the test to show that the code will work before you write the code. Testing and coding are performed together. Before any new software is released, the whole test suite is run to ensure that it does not break any other component of the system.
5. Some principles underlying the people aspects of XP are: work no more than 40 hours a week, be courageous, take responsibility and share ownership.
6. Some principles underlying code production are: keep it simple, have one shared metaphor to guide system development, regularly restructure the system to improve it (refactoring), continuously integrate and test, and follow coding standards.

III. RESEARCH DESIGN AND SURVEY CONDUCTED

In this section we conducted a research on three IT companies using both the qualitative and quantitative methodologies. As part of the qualitative method, data was collected through interviews and questionnaires. The quantitative data was collected by conducting a survey. The research was done with only three companies because of the time constraint and availability. Questionnaires were developed and were forwarded to the companies. Based on the filled questionnaires, we had conducted interviews on phone, or through e-mail. This Survey is designed to assess how far the development team followed XP practices. It is a subjective means of gathering adherence information from team members. The survey is composed of 15 questions on the extent to which each individual on a team uses XP practices (testing has been split to three categories and stand up meetings were added to the practices). A survey respondent self-reports the extent to which he or she used the practice, on a scale from 0% (never) to 100% (always). An overall score for the survey is computed via a average of each response of the three companies.

Adherence to XP practice = Σ (practice score * practice weight) /10 over 15 practices

Survey questions

For each question on table 1, the students were asked to use the following scale:

10: Fanatic (100%)
9: Always (90%)
8: Regular (80%)

7: Often (70%)
6: Usually (60%)
5: Half 'n Half (50%)
4: Common (40%)
3: Sometimes (30%)
2: Rarely (20%)
1: Hardly ever (10%)
0: Disagree with using this practice

Table 1: 15 survey questions

XP Practice	**Weight**	**Description / Question to be answered**
Automated Unit Tests	6%	You run automated unit test (such as JUnit,) each time you make a change. What % of your changes are tested with automated unit tests before they are checked in?
Customer Acceptance Tests	3%	Make sure both the developers and the customer know what they want. What % of your requirements have corresponding acceptance tests specified by the customer?
Test First Design	3%	Write test cases, then the code. The test case is the spec. What % of your code line items were written AFTER an automated test was developed for the corresponding scenario?
Pair Programming	9%	Two people, one computer. One thinks strategy, the other tactics. What percentage of your work (design, analysis, coding) was done in pairs?
Refactoring	9%	Rewrite code that 'smells bad' to improve future maintenance and flexibility without changing its behavior. What % of the time do you stop to cleanup code that has already been implemented without changing functionality?
Release Planning	6%	Customer and developers trade items in and out of the plan based on current priorities and costs. Adaptation is favored over following a plan.Do you allow for changes in release plans/requirements after each iteration based on customer feedback and current implementation?
Customer Access	6%	On Site Customer is best, you can use chat, etc. to quickly verify requirements and get feedback. What % of the time do you get quick interaction with your customers when needed?
Short Releases	6%	You have frequent smaller releases instead of larger, less frequent ones. This lets the customer see how it's going and lets you get feedback. How close are you to having releases that are about 3 months with interim iterations of a couple weeks?
StandUp Meeting	6%	The team takes 10 minutes each day to review what needs to be done each day and assigns user tasks to team members
Continuous Integration	9%	Code is checked in quickly to avoid code syncup / integration hassles. How often do you syncup and check in your code on average? (10 = 3 times a day, 8 = once a day)
Coding Standards	5%	Do you have and adhere to team coding standards? Besides brace placement, this may include things like logging and performance idioms. How often do you follow your team standards?
Collective Ownership	8%	You can change anyone's code and they can change yours. You don't get stuck when the expert is busy on vacation. People know many parts of the system. How often do people change code they did not originally write?
40-Hours Sustainable Pace	5%	People need to be effective over the long haul. How well do you pace yourself? Example Scores: 10 - I maintain a sustainable pace and the same high rate of output. 5 – I work longer than what I consider a sustainable pace, but still produce at a high rate and feel only a little burnt out. 2 – I work beyond a sustainable pace and feel burnt out. My code isn't at its usual high quality.
Simple Design	8%	Keep it simple at first; do the simplest thing that could possibly work. You don't follow the philosophy of "I'll include this because the customer might possible need it later" even though the feature isn't in the requirements. Also, you do not spend a lot of time on design documents. How often do you succeed in 'Keeping it Simple'?
Metaphor	6%	A single, overarching metaphor is used to describe the system. It is used by developers to help communicate ideas and to explain concepts to customers. How often do you feel this is true of the systems you develop?

Survey Results:

Results of the survey showed that the overall Adherence to XP practice was on the border of being acceptable in the range of 60 - 65%. At release level, the survey showed that there is an improvement, of the order 5%, from release 1 to release 3. This was attributed to the more experience employees got as they move from the first release to the last release. Feedback from employees showed that they have practiced some of the XP practices before even knowing about the existence of such methodology; in particular pair programming [5] that they used to practice within their graduation projects. Survey showed that they liked and are willing to work XP in their future developments. We quote here some of the comments we received from the employees:

"In general, we are very comfortable with XP methodology and we will use it in future projects, if nothing else is specified by user's non-functional requirements. The main reason is that it is code-oriented and leads to develop software without the upfront activity of detailed analysis, design and heavy documentation that are traditionally needed as for example in the waterfall methodology",

"In general, I like the XP and I would like to take another project that uses the XP model, in particular I liked the planning game and small size of documentation",

"The use of the XP in this project was good one, The project was small project, easy to manage, and easy to implement",and finally " The XP has some practices that we enjoy doing like game planning, small releases, incremental integration and sustainable pace".

IV. EXTREME PROGRAMMING LIFE CYCLE

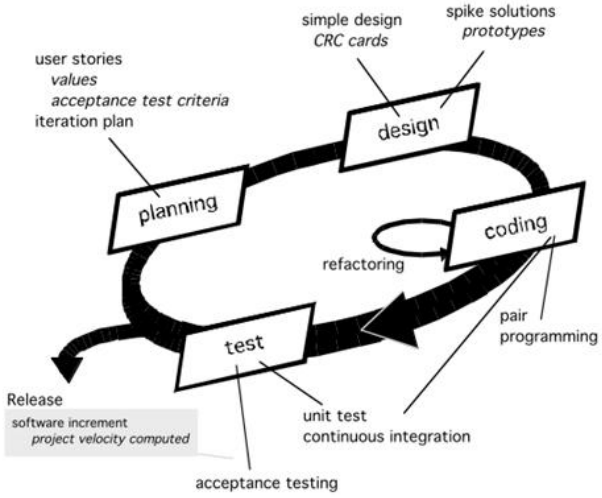

Fig 2: XP life cycle

There are five basic activities that XP proposes for software development process [12]

1. Planning

The first phase of extreme programming life cycle is planning, where customers or users meet with the development team to create 'user stories' or requirements. The development team converts user stories into iterations that cover a small part of the functionality or features required. The planning team prepares the plan, time and costs of carrying out the iterations, and individual developer sign up for iterations.

2. Designing

XP's simplicity principle doesn't mean that it can exclude designing process. Without proper design in the long run system becomes too complex and projects could come to a halt. It is then important to create a design structure that organizes the logic in the system so too many dependencies in the system can be avoided.

3. Coding

In XP coding is considered the only important product of the system development process. XP programmers start to generate codes at the very beginning so "At the end of the day, there has to be a program."

4. Testing

XP emphasizes to always check if a function works is by testing it. XP uses Unit Tests which are automated tests, and the programmer will write tests as many as possible to try to break the break the code he or she is writing.

5. Listening

Obviously, coding and testing need to be done no matter how a system is developed, but listening is very important in XP [4]. For XP developers the ability and expertise in technical aspects should be accompanied by the ability to be good listeners. This ability will enable them to understand what customers want and develop solutions which match customers' needs and desires as close as possible.

V. EXTREME PROGRAMMING PRACTICES

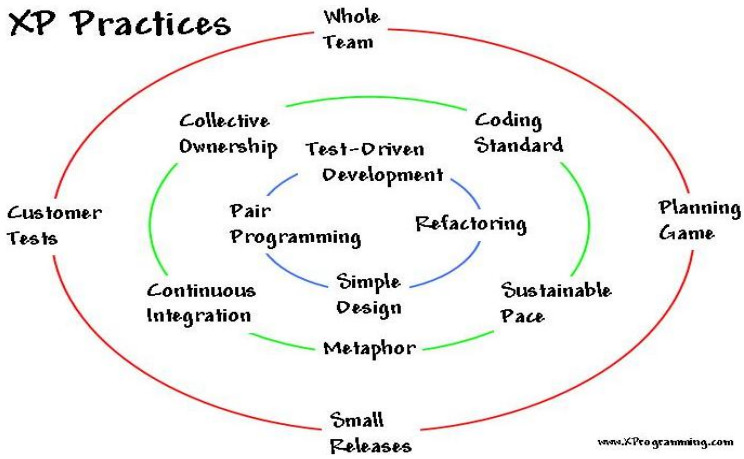

Fig 3: XP practices

The 12 XP activities are: [13]

1. Planning game: Quickly determine the next release's scope, combining business priorities and technical estimates. The customer decides scope, priority, and dates from a business perspective, whereas technical people estimate and track progress.
2. Small releases: Put a simple system into production quickly. Release new versions on a very short (two-week) cycle.
3. Metaphor: Guide all development with a simple, shared story of how the overall system works.
4. Simple design: Design as simply as possible at any given moment.
5. Testing: Developers continually write unit tests that must run flawlessly; customers write tests to demonstrate functions are finished. "Test, then code" means that a failed test case is an entry criterion for writing code.
6. Refactoring: Restructure the system without changing its behavior to remove duplication, improve communication, simplify, or add flexibility.
7. Pair programming: All production code is written by two programmers at one machine.
8. Collective ownership: Anyone can improve any system code anywhere at any time.
9. Continuous integration: Integrate and build the system many times a day (every time a task is finished). Continual regression testing prevents functionality regressions when requirements change.
10. 40-hour weeks: Work no more than 40 hours per week whenever possible; never work overtime two weeks in a row.
11. Onsite customer: Have an actual user on the team full-time to answer questions.
12. Coding standards: Have rules that emphasize communication throughout the code.

VI. EXTREME PROGRAMMING VALUES

Extreme Programming (XP) is based on values. The rules we just examined are the natural extension and consequence of maximizing our values. XP isn't really a set of rules but rather a way to work in harmony with your personal and corporate values. Start with five XP's values listed. [14]

1. 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.
2. 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.
3. 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.
4. 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.
5. 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 when ever they happen.

VII. BENEFITS OF EXTREME PROGRAMMING

Benefits of XP are [15]

1. For developers**,** XP allows you to focus on coding and avoid needless paperwork and meetings.
2. For the Customer, XP creates working software faster, and that software tends to have very few defects.
3. For management**,** XP delivers working software for less money, and the software is more likely to do what the end users actually want.
4. You can have small release of tested software at the end of each iteration with XP. The most important is that the software is visible to your customer at the end of every iteration.

5. Extreme Programming starts the simplest solution. You can add extra functionality afterwards. The purpose of this concept is to do planning, designing and coding for today, not for tomorrow.
6. The feedback from the system, customer and team are the key for success for extreme programming. Flaws in the system are detected in the earlier stage with this concept and customer can conduct acceptance test repeatedly to minimize error in your deliverables.

VIII. CHALLENGES OF EXTREME PROGRAMMING

Although XP methodology can result in an improved process which is more efficient more predictable more flexible and more fun it also has weaknesses such as: [16]
1. Difficulty coordinating larger teams
2. Can result in a never-ending project if not managed properly
3. Tendency to not document thoroughly
4. Predicting the precise features to be accomplished in a fixed time/budget

IX. CONCLUSION

In summary, there were several lessons learned and contributions from this paper. The lessons learned centered around the importance of one of the popular agile software development methodologies i.e. Extreme programming. 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. The Results also show that the overall Adherence to XP practice was on the border of being acceptable in the range of 60 - 65%.Survey showed that employees liked and are willing to work with XP in their future developments. We quote here some of the comments we received from the employees:
"In general, we are very comfortable with XP methodology and we will use it in future projects, if nothing else is specified by user's non-functional requirements. The main reason is that it is code-oriented and leads to develop software without the upfront activity of detailed analysis, design and heavy documentation that are traditionally needed as for example in the waterfall methodology",
"In general, I like the XP and I would like to take another project that uses the XP model, in particular I liked the planning game and small size of documentation",
"The use of the XP in this project was good one, The project was small project, easy to manage, and easy to implement",and finally " The XP has some practices that we enjoy doing like game planning, small releases,incremental integration and sustainable pace".
Also further research is needed for us to better understand the tradeoffs in risk and value between traditional and agile development using XP because very little little detailed reporting of the usage, penetration and challenges of extreme programming agile methodologies have been worked on.

REFERENCES

[1] A. Cockburn, Agile Software Development. Reading, Massachusetts: *Addison Wesley Longman, 2001.*
[2] K. Schwaber and M. Beedle, Agile *Software Development with SCRUM: Prentice-Hall, 2002.*
[3] A. Cockburn, Crystal Clear: *A Human-Powered Methodology for Small Teams: Addison Wesley, 2004.*
[4] K. Beck, Extreme Programming Explained: *Embrace Change, Second ed. Reading, Mass.: Addison- Wesley, 2005.*
[5] L. Williams and R. Kessler, *Pair Programming Illuminated. Reading, Massachusetts: Addison Wesley, 2003.*
[6] Raising, L., & Janoff, N.S. (2000). *The Scrum software development process for small teams. IEEE software,17(4), 26-32..*
[7] Boehm, B. (2002, January). *Get ready for agile methods with care. IEEE Computer, 35(1), 64-69.*
[8] S. Nerur and V. Balijepally, *"Theoretical Re -fl ections on Agile Development Methodolo-gies," Comm. ACM, vol.50, no. 3, 2007, pp. 79–83.*
[9] S. Nerur, R. Mahapatra, and G. Mangalaraj, *"Challenges of Migrating to Agile Methodolo-gies," Comm. ACM, vol.48, no. 5, 2005, pp. 72–78.*
[10] P. Schuh, *Integrating Agile Development in the Real World, Charles River Media, 2004.*
[11] http://www.agilemodeling.com/essays/proof.html
[12] https:// www.selectbs.com/glossary/what-is-extreme-programming.htm
[13] B. Rumpe, P. Scholz, *Scaling the Management of Extreme Programming Projects*, Projects & Profits Special Issue on Management of Extreme Programming Projects, Vol. III (8), pp. 11-18. ICFAI Press,Hyderabat, August 2003.
[14] https://www.unf.edu/~broggio/cen6016/ExtremeProgramming(XP)Article.pdf
[15] http://enterpriseblog.net/a/top-10-benefits-of-extreme-programming/
[16] 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 unknown.

