PAPER • OPEN ACCESS

A Systematic Review on Extreme Programming

To cite this article: Anchit Shrivastava *et al* 2021 *J. Phys.: Conf. Ser.* **1969** 012046

A Systematic Review on Extreme Programming

Anchit Shrivastava[1], Isha Jaggi[2,] , Nandita Katoch[3] , Deepali Gupta[4] and Sheifali Gupta[5]

[12345] Chitkara University Institute of Engineering and technology, Chitkara University, Rajpura, Punjab, 140401, India

*Corresponding Author: Deepali Gupta. Email: deepali.gupta@chitkara.edu.in

Abstract: Since the advent of start-ups and old companies starting to shift to e-business, it has become troublesome to choose the right methods for projects that's where extreme programming came to use. In this paper, we tried to compile different applications of the same. Extreme programming is a well-known agile method for software development. Extreme programming ensures customer satisfaction, better software quality and efficient project management. The team size is usually small but the group is team-oriented. It is a dynamic software development model, i.e., continuous discussion and integration of new features and ideas is the cornerstone of this model.

1. Introduction

Initially, the Waterfall model was used in which all the programmers would make a list of the requirements given by customers at once, then they matched up to their customer's requirements and delivered the product. But there were many problems in this model, customers frequently changed their mind and contradicted themselves. Sometimes, the customers were not sure what their requirements were. Even the programmers had their separate issues while working when they thought that the product was almost delivered, in reality, they would have covered only 1/3rd of the total route. Therefore customers and programmers demanded some iterable process which with shorter development cycles.
Kent Beck[1] along with Ward Cunningham and Ron Jeffries developed this methodology while he was working on Chrysler's comprehensive compensation system in 1996. He began to refine it and later wrote the book "Extreme Programming explained" in 1999, though he got out of the project in 2000.
As times are becoming modern, companies are shifting to more web based solutions. They usually outsource these tasks to small and young minds. These tasks demand faster delivery for which traditional software development approaches cannot be used. Therefore, programmers use popular agile techniques, such as Extreme Programming (XP), Crystal, SCRUM[13] and adaptive software development, to increase productivity without losing their quality of work. The main aim of using agile methodologies is to make an organization dynamic. All these Agile methodologies have some common characters such as continuous development, frequent customer consultation and small releases. A highly used agile methodology is XP.

XP is an agile software development framework that helps us produce high-quality software and also makes lives easy for the development team. It is usually practiced in small teams with up to 20 members and is really team oriented i.e. the product delivery is the joint responsibility of all the developers in the team rather than the owner or the boss of the company. This methodology takes this name as the traditional programming practices are taken to extreme levels. The basic purpose of developing this model was to create a light weight process model. When it comes to appropriate engineering practices in software engineering, XP is the most specific in the agile frameworks. It helps in creating the software according to the customer requirements. As it is an Agile software

development it consists of many releases in a short duration of time. It helps in improving the quality of software one step at a time. It promotes group development of a project in which programmers and customers are extensively involved. As they are developing the software customer can suggest changes and update as per the need, as the problem is better understood with time.
Extreme programming promotes location near to each other, ideally in a single room for more effective communication. Due to over hearing, the factor of hesitation can also be taken care of. It also provides close customer interaction thus it promotes that a representative of the customer should become an integral part of the team and communicates effectively. Most teams choose Extreme programming because it helps to reduce time loss while reading documentations as face-to-face communication is there. Developers can spend more time implementing their ideas rather than designing design ideas and preparing documentations. As long as the teams are small XP practices pay off as it is faster to share ideas by talking than to create documentations.

2. Characteristics of Extreme Programming (XP)

In Extreme Programming, minimum accompanying measures are required which means the need for creating documentation and project requirements is not required. It is also really team oriented, which means that it is a joint responsibility for all the developers to successfully complete the project and not just of the owner or boss of the team. XP is best practiced when the team size is small. Usually 12-14 people. It also promotes the involvement of user and customer at an very early stage, by this the time wasted by communication gap can be saved. It is also socially oriented.

3. Values and principles

There are 5 main values in XP methodology. They are as follows:-

3.1. Communication. Software development is all about understanding the need of the customer and implementing it. It is very important for team members to interact with each other. This method promotes the use of drawing tools for communicating [3].

3.2. Simplicity. It tries to make things as simple as possible by planning first. It helps in avoiding things that are not necessary and helps developers concentrate. It also promotes working on the present requirements and not predicting the future. The system design should also be kept simple for maintenance and improvement [4].

3.3. Feedback. Constant feedback from previous works can help them identify their areas of improvement and it also helps in making the design simpler.

3.4. Courage. If something you’re working on doesn't work it produces a lot of fear. In such times the previous principles should be kept in mind so that results do not harm the team.

3.5. Respect. All the members in a project i.e. the customer and the programmer should respect each other and accept feedback as it will help in making the project successful [5].

4. XP Practices

4.1. Customer Satisfaction. Designing really high tech and advanced products is irrelevant if it doesn't take care of what the customer wants [6]. To cope up with this XP follows two practices:

4.1.1. On-site Customer
Taking care of what a customer wants by acquiring the customer requirements and prioritizing them is the key to any successful business. Usually, they make a list of all the things that a customer wants and work accordingly. But, in the case of software development, it is difficult due to frequent changes in the requirements. Representatives from the customer side working along with the team can be really beneficial, as the questions can be answered without any delay and making assumptions by reading the customer requirements and speculating them can be avoided [7].

4.1.2. Small releases
As the requirements in software development change frequently, making small releases implementing the current customer requirements is beneficial.
The developers don't get confused due to too much work and the probability of making planning errors reduces significantly. Testing will be easier as there will be a few test cases to be taken care of making it a two weeks cycle, also it will be beneficial from a customer's perspective as they get to know the developers capability. Hence, they can choose if they want to continue giving work or not.

4.2. Software Quality. XP takes various measures to maintain software quality. It ensures that the growth is not compromised along with the necessary standards [8][19]:

4.2.1. *Metaphor*
A metaphor represents "What they are trying to achieve?" and explains to both the technical team and business. It serves as high level software architecture.

4.2.2. Testing
While developers write code, the customers should continuously test the product with various test cases (acceptance/functional testing) to ensure that it is properly functioning, and communicate the flaws. Even before writing the code, the developers write the test cases. They can also use test drivers after writing the code to check if everything is working perfectly, this can't be done with written documentation.

4.2.3. Simple Design
It is known that the requirements are continuously changing in case of software development, XP keeping this in mind promotes simple design and easy to understand code. It promotes a code with the least possible classes and functions/methods so that it can be understood with the least possible documentation outside the source code. By this the cost will not grow exponentially and also the developers can easily develop the code and understand it without needing to refer to the documentation.

4.2.4. Refactoring
All software get outdated with time. A business has to continuously change and improve in order to stay in business, being flexible enough to be changed is what will help the business grow. XP suggests that a code should not only be simple and easy to understand but also be flexible to be changed without changing the basic structure of the program or changing the functionality of the software [9][16].

4.2.5. Pair Programming
XP promotes the usage of pair programming, in this one programmer controls the keyboard and the mouse while the other gets time to think and plan strategically. All around the day programmers continuously swap their roles with others in regular intervals. Project managers often think that because of this costs will be increased. But it has been seen that if applied properly, only a 15-20% increase in effort is there. Not only the product quality increases and the time taken reduces, but it also results in the overall job satisfaction of the developers. Extreme programming does not mean that one programmer is typing and the other is watching, it is so that two minds can come up with a better solution [10].

4.2.6. Project Management

4.2.6.1. Planning game
Planning about the next release is really important. It suggests that it should be keep in mind the requirements put forward by the customer and the capabilities of the programmers. The things that need to be decided are [11]:

4.2.6.1.1. *Scope and Priority*
The features that are the most important, the features that need to be added further and the features that can be put off for some time.

4.2.6.1.2. Date and release date estimates
Two important questions to be taken into account are, when should the team release the next update and how much effort will be required by the team members.

4.2.6.1.3. Process and consequences
How will the team be organized and work and how will various deviations or challenges affect the team should be well prepared for smooth functioning during the time of crisis.
IET (Ideal Engineering Time) is used to estimate the effort in XP, which determines the tasks complexity and velocity (the team decides it by analyzing the previous tasks completed). Then the team has to define how many IET points they can use. For example, if last time they took 3 weeks, this time they will set the velocity to two weeks for the new release as well.

4.2.6.2. Sustainable Development
Working more than 40 hours a week will definitely lead to degradation in the work quality of a human. Extreme Programming suggests that the programmers should limit their working hours to a maximum of 40 hours a week, and over the time the streak should be restricted to two weeks. Kent Beck puts it perfectly as, "I want to be fresh and eager every morning, and tired and satisfied every night. On Friday, I want to be tired and satisfied enough that I feel good about two days to think about something other than work. Then on Monday, I want to come in full of fire and ideas." [10].

4.2.6.3. Collective Ownership
All the programmers in a project are collective owners of the source code. Anyone who can add value to the code at any instant is required to do so. There should not be divided responsibilities; the system should be such that anyone can improve any part at any time. The programmers will be able to get instant feedback on the code they wrote. Having automated test drivers will help the programmers to code with a free mind and without any fear.

4.2.6.4. Coding Standards
In a project many different programmers with different backgrounds will be there, it becomes really important that they understand each other's code, thus the code written should follow certain guidelines so that it is easily understandable and less time consuming.

4.2.6.5. Continuous Integration
It is required that the developers run their code several times a day, as they need to make sure that there is always an executable version of the program software. Integrating and building the system many times a day will be of help in functionality regressions, as and when requirements change. One good practice for continuous integration is, that it is possible to allocate a separate dedicated machine where programmers can take part to implement their code and undo changes if it didn't work.

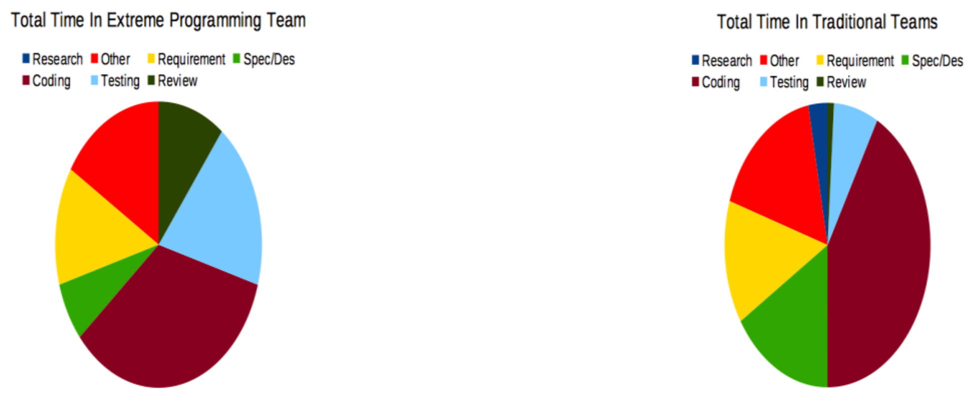

Figure 1: Time taken in XP teams[18]

Figure 2: Time taken in Traditional teams[18]

5. Distributed Extreme Programming

It may not be possible to practice Extreme Programming sometimes, due to unavoidable factors such as distance, unavailability of proper transport or the current pandemic situation. So, how can Extreme programming be applied in a distributed environment?

According to Michael Kircher et al. applying it in such situations is really easy, and is called distributed extreme programming. Though the practitioners may face various challenges such as making effective communication, having the proper infrastructure, availability due to different time zones or due to busy schedules and managing. The technology available today can easily take care of all of it, with something new coming up almost every day.

Distributed extreme programming is a lightweight methodology that has gained immense popularity. It promotes software development which depends on communication, simplicity, feedback and courage.

It is known that extreme programming (XP) has small teams and frequent changes in requirements, distributed programming promotes strong communication between team members and pair programming by use of various technologies available in the market.

Better communication not only increases understanding between the team members but also helps in the effective diffusion of skill set.

There are many advantages of practicing distributed extreme programming such as it will cost less because your investment in infrastructure will be reduced tremendously, customers or even the team can easily connect with each other without the need of travel, this will also help in saving time [12].

6. Extreme Programming from a Software Capability Maturity Model (CMM) Perspective

Software Capability Maturity Model (SW-CMM) is one of the six capability measuring models for software-defined by the Software Engineering Institute (SEI). According to SEI, it is an extensively used five-level model for structuring goals proficiently and describing engineering and management implementations up to the mark [14][20].

Below mentioned are the important characterizations of the five level model:

6.1. The Initial level. There aren't proper software management practices leading to chaos and instability. Very few operations are schematically formulated, and success entirely depends on individual effort or exceptional managers.

6.2. The Repeatable level. Bare minimum project management processes are structured to keep a check on revenue, managing time, and functionality. The mandatory process discipline is intact by making realistic plans with the aim to repeat previously acclaimed favorable outcomes.

6.3. The Defined level. This level assures the advancement and sustenance of the software method. The projects operate upon a pre-defined customized version of the organization's conventional software method by employing a combined set of precise software engineering and administration methods.

6.4. The Managed level. Quantitative goal setting is practiced at this level. Software processes and products are equally analyzed and controlled by quantitative measures.

6.5. The Optimizing level. Quantitative assessments and new ideas result in ceaseless improvement of the software process.

Even though the prime focus is on big projects and organizations, the same template can also be put in practice for divergent environments ranging from small start-up projects to mass projects with hard present and vital systems, by making minute changes.

In comparison, software subcontract management, which is applicable only for organizations that subcontract, the key process areas and goals of Extreme programming can be implemented anywhere.

Extreme programming keeps its focus on small-scale to moderate-sized teams, having software with indefinite or ever changing needs. The teams are almost collimated and have less than ten members.

Coding, testing, listening, and designing are the rudimentary tasks that make up Extreme programming.

7. A Lightweight process for E-Business Startups based on Extreme Programming (LIPE)

Since the start-up revolution in the tech industry, many investors eagerly invested in e-business with an aim to capitalize their ideas. These investors implemented traditional software development approaches. But oblivion to the ever-changing warfare of these start-ups and trying to follow the archaic methods led to poor management and difficulty to them. Later, they realised that these techniques didn't really cope up well and thus they had a hard time in the market. So, later on, LIPE was introduced which was a Lightweight Programming for E-business to ease their handling and marketing experience and create a base for the future.

Most of the software development organisations have shifted to e-business and also many start-up companies have emerged which led to decreased enthusiasm towards software engineering practices. But LIPE is considered to use all practices of extreme programming except for pair programming. To overcome the limitations of extreme programming, goal determined software measurement and inspections based on demand have been added. LIPE is interpreted from some inherent concepts of software process modelling [15].

After conducting various interviews with industry and on personal experiences on e-business software development, two major trends were observed, which are:

7.1. Current e-business software development keeps a sharp focus on upgrading legacy business-to-business systems such as EDI (Electronic Data Interchange) to operate over the Internet.

7.2. The focus is towards the implementation of new e-business solutions from scratch.
The first trend was mainly seen in older firms while the second was observed in start-ups. LIPE, also, focuses on the second one.

The business context of start-ups majorly depends on three factors:

- *Time-to-market.* A start-up has to exclusively depend on external capital for covering all sorts of costs before they start making revenue. The expenditure per month determines the time when more capital is needed. So the ability to generate revenue shifts this deadline more into the future and besides, adds up to the net present value. As a result, reducing launch time to market is a major concern for the start-up.

- *Size of the development team.* The teams are small in the initial stages. If the company begets a research institute, the team normally includes either young graduates or undergraduates, they can have both also.

- *Venture capital.* Often, venture capital and private or corporate angels finance the start-up organization. The main target of these companies is to have an initial show to public, within two to four years. During this, start-up companies go through various funding rounds and increasing the size of the development company. The start-ups need to show constant progress to stimulate the capitalists to fund them for the next round.

7.3. LIPE's Technical activities. LIPE's technical activities can be broadly described as four areas:
In Collect Scenarios, buyers and developers collectively develop the processes and customer notes down system usage scenarios also known as user stories by Extreme Programming. Scenarios have an important role in the procedure such as they state the required utility; accurately effort is calculated and regulated as per each scenario; finished scenarios mark each system development; priority-based iterations are planned etc.

In the Acceptance Test, the customer uses scenarios to check if the system requirements are fulfilled or not.

In real scenario, Refactor System and Rework Code, the developers write and test Java codes. But when looked at closely, they differ in terms of purpose and beginning points. In Realize Scenario, developers provide additional features that are provided as per each scenario, thereby extending the system.

In the Refactor System, rather than changing system functionalities, the internal design is reframed. Additionally, maintenance is improved.

In the Rework Code, defects described by open issue reports are fixed. In spite of having different purposes, the elementary steps conducted are identical in any of the three operations. The former makes sure that the code is readable and can be understood easily while the latter ensures that changes to the code are linked in such a way that issue-related metrics can be calculated easily.

8. Extreme Programming in large scale projects

The flexibility to make changes during the whole software development life cycle and a call to fast pace software development have led to the demand for lightweight or agile development software.These practices are effective for small and medium sized projects, in short with small teams. Extreme programming enthusiasts often assert that it is more profitable over standard methods, including cutting management expenses, increased team richness, and shorter deliverance rounds. Despite this, the relevance of agile strategies is stifled by determinants like size and kind of project, team members' skills, and engaged customers. Due to the dearth of up-front design and documentation, directly adopting large, complex projects is too arduous a task.

In spite of that, it is acknowledged by most specialists that agile practices and conventional strategies are, in a way, congenial to one another. For instance, the SWCMM model is one such method that deploys Extreme programming and is usually employed for extensive projects.

Tremendous efforts have been made to tailor extreme programming practices for big, intricate projects.

Majorly three problems are faced by complex systems:

8.1. A thin spread of application domain knowledge

Deep-application data from the main development team, needed by extensive projects, is minutely spread within several software-development team staff. There is a need for an ample amount of effort for maintaining a general understanding of both the application domain and the functioning of the system's performance.

8.2. Fluctuating and conflicting requirements

Inadequate application knowledge of developers and situations such as switching business aims may produce wavering and conflicting demands in large projects.

8.3. Communication and coordination breakdowns

Software development requires the coordination of a large number of groups to share information. Agile technologies depend primarily on the implicit experience of team members. These practices involve high risks. While dealing with a huge number of stakeholders and a magnanimous chunk of data, informal communication is not efficacious. Standard procedures with fixed goals decrease risks by deploying several tactics like proper documentation, regular evaluation, etc.

Active change in project demands and the pressure for time to put the product on sale is also faced by big complex projects. To decrease the risks, extreme programming has started being adopted in large-scale projects. But, the received outcomes were mixed. As per some studies, some agile practices like regular testing, small deliverance, and refactoring, are appropriate for massive projects. On the other hand, standing-meetings and employing tropes to outline system structures are improper.

9. Tailoring Extreme Programming for Larger and Complex projects

XP is a really helpful software engineering practice which makes the teams really productive. Though it is only for small teams, it can also applied to large and complex teams just by tailoring a little.

9.1. Designing up-front. Extreme programming tends to understate up-front design as it insists that everything is evolving. Alternatively the fundamental parts along with correlations of the application are described by a 'metaphor'. Yet, for complex projects, structural design in advance is acknowledged to be crucial as it lessens the development period for unique functionalities. The structural depictions made beforehand also cut developers' priming time thereby alleviating the expenses of bringing someone fresh onboard. It also helps control the costs of the replacement and developers yield a more lucid perception of the complete setup of institution.

9.2. Short release cycles with a layered approach. Sequences with releases of brief periods and constant incorporation together are considered amongst the salient attributes of agile practices similar to Extreme programming which garners the spotlight. The layered method, similar to extreme programming's short release methods, delivers end-to-end features that can operate as a result after every brief repetition. The difference between this and extreme programming is that span of the repetitions is not established though it depends upon the kind of tasks.

9.3. Surrogate customer engagement. A genuine onsite customer with proper knowledge and decision-making capability is the ideal client for extreme programming. The primary determinant for any extreme programming project's success is engagement with the customer.

But, in actuality, finding access to such customers is tricky. In large-scale projects, The difficulty further magnifies in large projects where the complicated structure of the reinforcement field usually exceeds the skill or proficiency of some customers and developers. Additionally, oftentimes the right customers are not end-users. There are product managers or business analysts in immediate contact with the customer. There are regular meetings within the development team and product managers to discuss various necessary changes almost every day. Practices like this lead to a more reliable project result in comparison to projects with sparse client engagement.

9.4. Flexible pair programming. Though a good practice, pair programming is not viewed as practical in all circumstances. In software development, very few things are done in pairs. The underlying motto for flexible pairing is to motivate the developers and keep their spirits up.. Sharing desktops is a pliant aspect for the developers as well.

9.5. Identifying and managing developers. A developers' understanding of scheme patterns and the ability to provide standard interfaces is critical for a successful project. The developer's morale should be upheld well for better results in projects. The chief factors affecting developer's zeal are adjustable working hours, remote working and a result-focused environment.

9.6. Reuse with forward refactoring. Reuse is facilitated by a pattern-based approach and refactoring magnifies reuse over functionalities. But, refactoring may take on a different form as compared to extreme programming practices.

9.7. Controlled empowerment -organizational structure. By means of standard interfaces, the empowerment is controlled elegantly. Software developers need to use a standard approach for solving the problem given to them [9].

10. Drawbacks of Extreme Programming

Like other software engineering practices, XP also has some drawbacks also, they are listed below:

10.1. Code overcomes design. Extreme programming centers around code. The design is the main USP of any software or application, so this could go wary against the customer's wishes if the design is not at par with their requirements. Additionally, it may compromise the implementation of software specifications.

10.2. Location. Implementing extreme programming restricts the scope of projects. The reason behind this is that Extreme programming interactions work well when team members meet in person as the project involved is cumbersome to employ without the presence of the customer.

10.3. Lack of documentation. Documentation reduces the recurrence of errors. In extreme programming, continuous changes could not be documented well. Hence, tracking unforeseen failures is difficult and risk is often quite high.

10.4. Stress. Software quality can decrease if the developers are stressed. Meeting with fixed deadlines and high pressure may result in errors while coding [17].

11. How is Extreme programming different from other Software Engineering practices

Choosing the right software engineering practice for your project is really important, below is XP compared to other popular software engineering practices:

11.1. Comparison between Extreme Programming and SCRUM. Scrum is a type of Agile framework where people can address a complex problem keeping in mind the high quality, productivity and creativity using an iterative process called sprints. Refer to **Table 1** for better clarification [21].

Table 1. XP vs SCRUM

Extreme Programming	SCRUM
The team works for 1-2 weeks	The team works iteratively, known as sprints. Usually 1-2 month long sprints.
Has a flexible timeline.	Does not allow any changes in timeline or guidelines.
Emphasizes on Software Engineering methodologies.	Emphasizes on self-organization.
Team has to strictly follow the predetermined priority order.	The team decided the order of the priority.
Ready to apply without any changes.	Not complete, will have to fill the framework with any other software engineering technique.

11.2. Comparison between Extreme Programming and Agile. Agile was developed in early 2000's to improve the product quality. Unlike traditional approaches, it has extensive customer engagement and self-managing teams.

As XP is also an agile method, they both have these common features. Unlike, agile methods, there is no intermediary between the customer and the development team. The project is divided into small parts and each small part's plan can be changed as and when required.

11.3. Comparison between Extreme Programming and Waterfall. Before Agile methods were introduced, all the projects were managed with the waterfall methodology. There are a lot of differences between the two such as, XP is iterative unlike the waterfall model in which the same lifecycle is followed till the project is created. Also, in XP customers are constantly involved with the development team, suggesting changes as and when required. XP is basically Agile team without strict framework.

11.4. Comparison between Extreme Programming and Kanban. The main difference between the two is that in Kanban the workflow in not divided in small parts.

11.5. Comparison between Extreme Programming and Lean. Lean believes in developing a small part of the project only when the customer asks the team to develop [22].

12. Conclusion

Extreme programming is an extremely time-efficient and lightweight software development technique based on principles of ingenuousness, interaction, feedback and strength. Such methodologies have gained extensive popularity due to the need for fast-paced software development and the ability to accommodate changes. Teams using Extreme programming deliver software with low fault rates. Also, each technical hurdle is not an obstacle but rather a medium to acquire skills. Inculcating extreme programming principles in companies creates a motivating and compelling atmosphere within and between teams.

From this paper, it was deduced that extreme programming and similar agile methods are highly reliant on size and kind of project, skilled team members and additionally, customer engagement.

Extreme programming focuses on individuals and is such a model that readjusts well to dubious or unpredictable demands.

It depends heavily on the joint effort between developers, programmers and clients. This model works efficiently in changeable situations.

13. References

[1] https://en.wikipedia.org/wiki/Extreme_programming

[2] Dietm Pfahl 2014 *Software Engineering* https://courses.cs.ut.ee/MTAT.03.094/2015_fall/uploads/Main/SE2014-handout11.pdf

[3] A. van Deursen 2001 *Program comprehension risks and opportunities in extreme programming* (IEEE) pp. 176-185.

[4] Don Wells 2013 *Extreme Programming : A gentle introduction* http://www.extremeprogramming.org

[5] K. Beck 2000 *Extreme Programming Explained: Embrace Change*

[6] Mohammadi S, Sohrabi S, LinLin Z and Youcong in 2009 *Challenges of user Involvement in Extreme Programming projects* (International Journal of Software Engineering and Its Applications Vol. 3, No. 1.)

[7] Newkirk J 2002 *Introduction to agile processes and extreme programming* (IEEE) pp 695-696

[8] *Extreme Programming*. https://www.agilealliance.org/glossary/xp/#q=

[9] Cao L, Mohan K, Peng X and Ramesh B 2004 *How extreme does extreme programming have to be? Adapting XP practices to large-scale projects* (IEEE) pp 1-9

[10] Beck K 2006 *Extreme programming: A humanistic discipline of software development* (International Conference on Fundamental Approaches to Software Engineering) pp 1-6

[11] Maurer F, Martel S 2002 *Extreme programming. Rapid development for Web-based applications* (IEEE) pp 86-90

[12] Kircher M, Jain P, Corsaro A and Levine D 2001 *Distributed eXtreme Programming* (Second international conference on eXtreme Programming and Agile Processes in Software Engineering) pp 66-71

[13] Anwer F, Aftab S, Shah S and Waheed U 2017 *Comparitive Analysis of two Popular Agile process models: Extreme Programming and Scrum* (IJCST Volume 8) pp 1-7

[14] Paulk M, Curtis B, Chrissis M and Weber C 1993 *Capability Maturity Model for Software* (Version 1.1)

[15] Zettel J, Maurer F,Münch J and Wong L 2001 *LIPE: A Lightweight Process for E-Business Startup Companies Based on Extreme Programming* (Product Focused Software Process Improvement, Third International Conference) pp 255-270

[16] Cao L, Mohan K, Peng X and Ramesh B 2004 *How extreme does extreme programming have to be? Adapting XP practices to large-scale projects* (IEEE) pp 1-10

[17] Panayotova E 2018 *What are the pros and cons of extreme programming* https://simpleprogrammer.com/pros-cons-extreme-programming-xp/

[18] Macias, Holcombe and Gheorghe 2003 *A Formal Experiment Comparing Extreme Programming with Traditional Software Construction* (International conference on Computer Science) pp 6-17

[19] Lindstrom L,Jeffries R 2004 *Extreme Programming and Agile Software Development Methodologies. (*INFORMATION SYSTEMS MANAGEMENT SUMMER) pp 41-52

[20] Paulk M 2001 *Extreme Programming from a CMM Perspective* (IEEE Software) pp 19-26

[21] Difference between SCRUM and XP 2019 https://www.geeksforgeeks.org/difference-between-scrum-and-xp/

[22] Sergeev A 2016 *Extreme Programming, Waterfall, Agile, Kanban, Scrum, Lean- What's the difference?.* https://hygger.io/blog/extreme-programming-waterfall-agile-kanban-scrum-lean/#:~:text=Extreme%20Programming%20and%20Agile,-The%20most%20general&text=Like%20other%20Agile%20methodologies%2C%20Extreme,is%20subdivided%20into%20small%20sections.

Acknowledgement
This research was supported by Chitkara University, Punjab. We thank our mentor Kulbhushan Sharma who provided insight and expertise that greatly assisted the research, for comments that greatly improved the manuscript.

We would also like to show our gratitude to Dr. Rajnish Sharma, Chitkara University for sharing his pearls of wisdom with us during the course of this research, although any errors are our own and should not tarnish the reputations of these esteemed persons.