Journal of Innovative Engineering and Research (JIER)
Vol.- 6, Issue -2, October 2023, pp. 16-19 (4 pages)

Extreme Programming in Software Development

Dr. Sanjay Singh Bhadoriya[1] and Saurabh Parikh[2]

[1,2] *Department of Computer Application, Dr. A. P. J. Abdul Kalam University, Indore 452010, India*
Corresponding Author Email: saurabhmparikh@gmail.com

*Abstract— **In recent years, the practice of Software Development (SD) has become more common in the software industry. The advantages of globalization, the economy, location, speed to market, organizational strategy, the availability of qualified workers, and lower prices are all factors in this shift. One of the most well-known agile approaches is Extreme Programming (XP), which places a premium on clarity, openness, bravery, and feedback. High demand and a need for goods from customers and clients are putting a strain on the software industry's ability to provide high-quality items quickly. Small delays in the release may have a huge influence on the company's image and profitability. In software engineering, cost estimate plays a significant role in contract negotiation and project execution. A good Development plan reduces the risks and the inefficiencies related to the Development task. Development effort and productivity are difficult to quantify because of a variety of internal organizational elements and procedures related to the product itself. Development process productivity is often lower than development process productivity.***

Index terms: **Extreme Programming, Software, Development, Technology.**

I. INTRODUCTION

Today's world is entirely influenced by SW technology. There is almost no one left who doesn't use some kind of SW technology on a daily basis now. These SW components may be accessed in a variety of ways, including standalone programs, web programming, mobile apps, and more. As the usage of SW technology grows, so does the need for new, improved, and high-quality SW products. Nowadays, practically everyone is linked to technology in some way, whether directly or indirectly, thanks to the widespread availability of widely used mobile phones.

Government, as well as community and private efforts, have all been impacted by the expansion of technology. Without technology, we are unable to confront the world and lead regular lives. As may be seen from the above, there are a large number of users of SW. As the number of users grows, so does the need for innovative and high-quality software. The process of creating software is one of iterative, resource-constrained, well-focused development. In order to entice people to utilize technology, new and improved software (SW) must be developed. It's a must for the industry's success. High demand and a need for goods from customers and clients are putting a strain on the software industry's ability to provide high-quality items quickly. Small delays in the release may have a huge influence on the company's image and profitability. This situation is a result of companies still adhering to the traditional software development process. Until recently, the process of developing software was seen as a sequence of processes that were followed by the implementation of code. While it may have worked well at first, as use grew, it became less appealing to both the client and the end user. Increased consumer demand has resulted in an increase in competition for the best possible product.

II. LITERATURE REVIEW

Hohl, P., Klünder, J., van Bennekum, A. et al. (2018), In 2001, seventeen professionals set up the manifesto for agile software development. They wanted to define values and basic principles for better software development. On top of being brought into focus, the manifesto has been widely adopted by developers, in software-developing organizations and outside the world of IT. Agile principles and their implementation in practice have paved the way for radical new and innovative ways of software and product development. In parallel, the understanding of the manifesto's underlying principles evolved over time. This, in turn, may affect current and future applications of agile principles. This article presents results from a survey and an interview study in collaboration with the original contributors of the manifesto for agile software development. Furthermore, it comprises the results from a workshop with one of the original authors. This publication focuses on the origins of the manifesto, the contributors' views from today's perspective, and their outlook on future directions. We evaluated 11 responses from the survey and 14 interviews to understand the viewpoint of the contributors. They emphasize that agile methods need to be carefully selected and agile should not be seen as a silver bullet. They underline the importance of considering the variety of different practices and methods that had an influence on the manifesto. Furthermore, they mention that people should question their current understanding of "agile" and recommend reconsidering the core ideas of the manifesto.

Theo, The unissen, Uwevan Heescha, Paris Avgeriou (2022) With an increase in Agile, Lean, and DevOps software methodologies over the last years (collectively referred to as

Continuous Software Development (CSD)), we have observed that documentation is often poor. Objective: This work aims at collecting studies on documentation challenges, documentation practices, and tools that can support documentation in CSD. Method: A systematic mapping study was conducted to identify and analyze research on documentation in CSD, covering publications between 2001 and 2019. Results: A total of 63 studies were selected. We found 40 studies related to documentation practices and challenges, and 23 studies related to tools used in CSD. The challenges include: informal documentation is hard to understand, documentation is considered as waste, productivity is measured by working software only, documentation is out-of-sync with the software and there is a short-term focus. The practices include: non-written and informal communication, the usage of development artifacts for documentation, and the use of architecture frameworks. We also made an inventory of numerous tools that can be used for documentation purposes in CSD. Overall, we recommend the usage of executable documentation, modern tools and technologies to retrieve information and transform it into documentation, and the practice of minimal documentation upfront combined with detailed design for knowledge transfer afterwards. Conclusion: It is of paramount importance to increase the quantity and quality of documentation in CSD. While this remains challenging, practitioners will benefit from applying the identified practices and tools in order to mitigate the stated challenges.

Shrivastava, et al (2021) Extreme programming was developed as a solution to the problem of method selection that has arisen with the rise of start-ups and the transition of established businesses to online commerce. In this study, we aimed to collect a variety of relevant examples. One of the most well-known agile approaches to developing software is called extreme programming. Customer happiness, improved software quality, and effective project management are all benefits of extreme programming. Teams tend to be small, but everyone works well together. This paradigm is based on ongoing dialogue and the incorporation of new ideas and features, making it a dynamic approach to creating software.

Maleeha, Yasvi (2019) Extreme programming is an iterative approach to software development that attempts to make better software and aid in finding the best possible solution. Extreme Programming is distinct from other approaches to software development because of its emphasis on flexibility and rapid response to evolving client needs. Better outcomes have been achieved in software development thanks to the use of extreme programming as a technique.

Sadath, Lipsa & Karim, Kayvan & Gill, Stephen (2018) Developing software that is both functional and easily maintained in order to fulfill the needs of a certain use case is an example of software engineering. Due to the higher degree of abstraction required for this kind of manufacturing, it differs fundamentally from other forms of engineering practice. This reality gives rise to several strategies for long-term success in the software business. This method's significance in ensuring the long-term health of the academic software sector cannot be overstated. In order to establish a shared understanding and technical foundation in the academic community, we propose a framework called XPIA (Extreme Programming In Academia) that employs tried-and-true methods from the software engineering industry, with a special emphasis on pair programming.

III. SOFTWARE DEVELOPMENT

A. It is expected that software will evolve and alter throughout the course of its lifespan. A software product's lifespan necessitates development. For the sake of keeping the program running, preventing and correcting software errors, and enhancing its usefulness, software development is necessary. Developing software is the process of making changes to a software system or a component after it has been delivered in order to fix bugs, enhance performance or other characteristics, or to adapt to changing environmental conditions. Corrective, adaptive, perfect, and preventative are some of the classifications of Development. Adaptive Development focuses on adapting to changes in the software environment, whereas perfective Development focuses on meeting new user needs. Error correction is under corrective development, while error prevention falls under preventive development, which aims to keep problems from occurring in the first place. It has been noticed that corrective development is believed to be conventional, while others are considered to be a kind of evolution in software. Most change requests in Development fall into the corrective or perfective category, according to recent research.

B. Development accounts for between 40 and 90 percent of the overall cost of a software system's lifespan, according to many empirical studies. There is always a need for new software systems to stay up with developments in the software environment. It is more cost-effective to reuse and improve existing systems rather than to develop a new one, thanks to the economic advantages. The current techniques and models of development are taken from the current development approaches and models. New software development paradigms have emerged as a result of the dynamic and demanding nature of the software industry.

Software development is often referred to be an iterative process because of the way development is conducted. Examining these distinctions more closely demonstrates how Development differs from the software development life cycle. It is also essential to have a Development-conscious model since the beginning phases of Development are more critical and need more work than development. We have our own set of procedures for creating computer programs called "software development life cycles". It's important to follow these models since they break down the development process into discrete steps that must be completed in the correct order. Quick repair, Boehm, Osborne, iterative-enhancement, full-reuse, etc. are all examples of development-conscious approaches. Traditional software development process models, which accept change requests as input and conduct all stages, are the basis for these process models.

IV. Development Estimation

In order for software development to be a success, accurate estimations are critical. Despite the fact that software development is a significant activity and a significant portion of the overall cost of software, researchers devote less attention to software development estimate than they do to software development estimation for new products. Software development has fewer methods for estimating work than software development. Models such as the ACT model, the FP model, and COCOMO 2.0 reuse may estimate software development effort. Source lines of code, function points, and object points are used as scaling units in many estimate techniques. The current software development estimating methods are based on the old software development approaches. Realistic results may be achieved using these models since they take into account the size of a program in terms of function points and source lines of code, which are not appropriate metrics for extreme programming.

Changes in Legacy Code: Changes to existing code, such as fixing bugs or adding new features, are at the heart of development. Code changes are a time-consuming and costly undertaking because of the absence of test coverage, outdated documentation, and a lack of access to original programmers. Because of its complicated structure, it's difficult to estimate the effects of modifications to old code. As a consequence, system instability and defects may emerge from code modifications that have insufficient test coverage.

Impact of XP practices on maintainability during Development: The maintainability of a software product has a significant impact on both the maintenance costs and the product's overall lifespan. Non-XP software produced using antiquated and unstructured code has a significant maintainability concern. Because of this, an experiment is needed to see how an iterative development life cycle utilizing XP affects maintainability, productivity and other aspects of Development. XP (or individual practices) has been the subject of several evaluation studies, both by industry professionals and by students in project courses, to determine whether or not they have an impact on development quality and productivity.

V. EXTREME PROGRAMMING

Extreme Programming (XP) is one of the numerous prominent agile approaches. It was created in 1996 by Kent Beck, Ward Cunningham, and Ron Jeffries, and released in 1999. The first time XP was used to update the Daimler-Chrysler payment system was on the Chrysler Comprehensive Compensation (C3) project. XP's incremental development technique is best suited for environments that are constantly changing. It is the goal of XP to increase the quality and responsiveness of software to changing client needs. Extensively used procedures like as code reviews and testing have been adapted for usage in XP. These procedures are applied to an extraordinary degree. XP is organized on a philosophy with five values and a set of twelve practices. The principles that are important to XP include communication, simplicity, feedback, bravery and respect. It's important to remember that these values are interdependent and complement each other. Communication and feedback, for example, aid in establishing a shared understanding of the project's goals and progress.

- **Small Releases**

XP requires a two-week cycle. Every two weeks, some new feature must be made available. In a few of months, the system will be in full operation. Every day or every other month, a new release is created. XP's iterative character is underscored by its small release strategy. Since XP was released so quickly, its feature set is quite limited. Customers benefit from a sense of security about the project's development because of these frequent, short releases. Developers have validated the functionality of deployable software at the conclusion of each iteration in preparation for showing it to clients. For the end user, the customer may select when and how the product is released. Small functional units that make good business sense and can be released into customer environments are the focus of the release planning.

- **Metaphor**

An example of this is the use of a metaphor, which is a short story about how a system works. A metaphor or combination of metaphors shared by consumers and programmers defines the system's form. The goal of metaphor is to create an architecture that is simple to communicate and elaborate between the client and the developer.

- **Sample Design**

KISS (Keep It Simple, Stupid) is a philosophy advocated by the XP methodology, which states that the development team should strive to keep the system as simple as possible. To ensure that every time the test runs, everything is communicated exactly as desired by programmers, the code must be concise and free of duplicate code.

- **Tests**

XP places a high value on testing and test-driven development. It necessitates the creation of unit tests prior to the actual authoring of code. Before developing a single line of code, programmers write unit test cases. Minute by minute, unit tests are written. Tests written by customers guarantee that features are performed as planned. Refactoring requires the usage of unit tests in order to get immediate feedback from the system, which is why they are an essential part of the process.

- **Refactoring**

Small, quick changes are made to the system without affecting its behavior as part of the refactoring process. Duplicate code is removed, communication is improved, the process is simplified, or more flexibility is added so that all tests continue to execute. A system's exterior behavior isn't altered, but its extensibility and readability are. The simplicity of refactoring allows you to experiment with other designs. Code modifications need a strong sense of self-confidence. It is true that refactoring streamlines designs. The refactoring process facilitates the simplification of code in order to make it more general.

- **Pair Programming**

Pair programming is a kind of programming in which two programmers work together on the same computer system to produce code. Driver and observer are the two roles that are defined in this XP methodology. The observer evaluates the code written by the driver.

VI. CONCLUSION

To ensure excellent software is delivered on schedule with little effort, several approaches have been created to impose a disciplined procedure on the creation of software. In response to the problems with traditional approaches, "agile techniques" have emerged. XP is an early and widely used form of agile development. Many in business and academics have hailed XP as revolutionary, yet there is few quantitative evidence to back up their assertions. The goal of this study was to determine whether or not XP increased software production, decreased the cost of change, and improved the efficiency with which new developers picked up skills via pair programming. Software development is often referred to be an iterative process because of the way development is conducted. Examining these distinctions more closely demonstrates how Development differs from the software development life cycle.

REFERENCES

1. Hohl, P., Klünder, J., van Bennekum, A. et al. Back to the future: origins and directions of the “Agile Manifesto” – views of the originators. J Softw Eng Res Dev 6, 15, 2018.
2. Theunissen, Theo & Heesch, Uwe & Avgeriou, Paris. A mapping study on documentation in Continuous Software Development. Information and Software Technology. 142, 2022 106733. 10.1016/j.infsof.2021.106733.
3. Shrivastava, Anchit & Jaggi, Isha & Katoch, Nandita & Deepali, Gupta & Gupta, Sheifali. A Systematic Review on Extreme Programming. Journal of Physics: Conference Series. 1969, 2021.
4. Yasvi, Maleeha. Review On Extreme Programming-XP, 2019
5. Sadath, Lipsa & Karim, Kayvan & Gill, Stephen. Extreme programming implementation in academia for software engineering sustainability, (2018). 10.1109/ICASET.2018.8376925.
6. Darwish, N. R. Enhancements in Scum Framework Using Extreme Programming Practices. International Journal of Intelligent Computing and Information Sciences (IJICIS), Ain Shams University, 14(2), 53-67, 2014
7. Abdullah, Elmuntasir & Abdelsatir, El-Tigani. Extreme programming applied in a large-scale distributed system. 442-446, 2013 10.1109/ICCEEE.2013.6633979.
8. Shrivastava, Anchit & Jaggi, Isha & Katoch, Nandita & Deepali, Gupta & Gupta, Sheifali. A Systematic Review on Extreme Programming. Journal of Physics: Conference Series. 1969. 012046, 2021, 10.1088/1742-6596/1969/1/012046.
9. Rojas, S. & Guzmán, L. & Coronel, P. & Benítez, A. Scrum with eXtreme Programming: An Agile Alternative in Software Development, 2021, 10.1007/978-3-030-60467-7_29.
10. Ekhlaif, M. & Elshaar, S.A. A systematic study of extreme programming and their implementation in Libyan Software. 23. 410-417, 2013, 10.5829/idosi.wasj.2013.23.03.13071,