HBRP PUBLICATION

Recent Trends in Information Technology and its Application
Volume 4 Issue 1

The Best Practices of Extreme Programming (XP) Quality (Review)

***Rana Alaudeen Abdulrahman*[1], *Sawsan Ali Hamid*[2*], *Dr. Ruaa Ali Khamees*[3]**
[1,3]*Institute of Technology, Middle Technical University, Baghdad, Iraq.*
[2]*College of Computer Science, Tikrit University, Tikrit, Iraq.*

[*]***Corresponding Author***
E-mail Id:-*sawsan.ali@tu.edu.iq*

ABSTRACT

Software engineering (SE) plays an important role for improving society's well-being through the use of high quality software. There is noted that most of the software projects are failed, due to missing or poor software development practices in software organizations. Due to this reason, having a good and sound software development methodology is crucial for software organization to satisfy stakeholder's requirements. One of the prevalent software development methodologies in SE is Extreme programming (XP) methodology. As a matter of fact, an appropriate software development methodology is a fundamental to reach stakeholders' satisfaction. Within context, it has been a notable failure in software development projects due to the frailty usage of software development methodologies in software organizations. However, Extreme Programming (XP) is an emerging software development methodology that affects positively in term of quality, time and cost among other methodologies.

Keywords:-*Software engineering, agile software, Extreme programming (XP)*

INTRODUCTION

Software engineering (SE) is an area that deals with the field of software construction engineering. As guidance in software development, it has been held systematic and has practical methodologies. The software life cycle consisting of the elicitation and review of requirements, design specifications, implementation, verification and validation, deployment and maintenance has been demonstrated [1]. Software development processes are an important part of software engineering, which influence the product outcome [2,3].

Nowadays, business processes are more complex, interconnected, interdependent, and interrelated than ever before. Due to this multifaceted nature of businesses, the software industry is strongly going toward the use of the methodologies which have been developed from practices such as agile methods[4,5]. Recently, the agile methodologies family – such as Extreme Programming (XP), Scrum, and Adaptive Software (ASD), have become extremely established in software engineering. In general, agile is characterized by the following attributes: incremental, cooperative, straight forward, and adaptive. However, agile methods are iterative processes, where stakeholders and developers work together effectively, understand the system's idea, identify the requirements, and prioritize the functions of the system[6]. Additionally, agile software methods emphasize on delivering the software after iteration. They emerged as a response to the inability of previous plan driven approaches to handle rapidly changing environments[7].

SOFTWARE DEVELOPMENT PRACTICES METHODOLOGY

Software engineering methods often introduce a new set of criteria for software

quality and a special language-oriented or graphical notation[8,9]. A notation is a system of characters, symbols or abbreviated expressions used to express technical facts or quantities and usually a technique uses a notation [10]. For example, structured analysis and design, object-oriented analysis and design and prototyping are methods. Techniques of structured analysis and design are for instance data flow diagrams and entity-relationship diagrams that can be described by using annotation. Paradigm the term (software engineering) paradigm is often used to refer to a set of steps that consist of methods, tools and procedures [11,12]. A paradigm is also used in order to perceive the different phases in development. Phases are decomposed into tasks and activities and tools such as templates, forms and checklists are used to complete the tasks and activities [13,14].

Software engineering approaches from part of a quality assurance system, and may include methods such as waterfall, prototyping, iterative and incremental development, spiral development, rapid application development, and extreme programming [15-17]. Thus, study the software development methodologies and their stages is essential in improving the software industry. The software development process, along with its associated systems analysis and design phase, needs to be more adaptive as the business community advances into the future economy [18-21]. The process of software development has progressed through three significant historical stages, including (1) developer-as-artist, (2) developer-as-engineer, and (3) agile methodologies [22-24].

According to Valacich, George, and Hoffer [22] the first of these phases in software development, developer-as-artist, was evidenced by software developers not documenting the programs being developed or not utilizing automated tools during the development process. The software developers in this phase were considered geniuses and artists as a high degree of dependence on the software developer was necessary for continued maintenance. The next phase, developer-as-engineer, was when organizations brought more control and regulation to the software development arena as the development process and the lifecycle of software development became a more structured process[22]. This is where the rise of a waterfall system development methodology was formed, in which the system development lifecycle is more of a linear process and moves in strict order from the actual software system concept through the software system design, implementation, testing, installation, and troubleshooting, and finally ends up with the ultimate operation and maintenance of the software system [17,24].The rise of the third phase, agile development methodologies, has been ushered in over the last few years as the growth of the Internet economy and object-oriented approaches have intersected [22].

According to Leffingwell [25] there are several methodologies was developed by the developers, one of the main software development methodologies is an agile methodology. Agile software development methodologies require closer cooperation between programmers and the ultimate business user community that will combine a number of software lifecycle phases into fewer phases, and involve multiple iterations of software implementations within an application system [26-28]. Prototyping, time constraints, smaller project team members, management involvement, and iterative software development are all significant components of the agile software development process[29,30]. This new concept of agile software development has aided in adding value to software

generation and seems to fit into a world where the requirements for businesses to develop application software are at a faster pace to meet the demands of a changing environment [26,31].

AGILE SOFTWARE DEVELOPMENT

Agile software development is an approach to software development that, in addition to programming, concentrates on subjects like project management and teamwork. Agile is a philosophy or a way of thinking about software development and there is no single unified agile methodology to follow [25,32-34]. The term agile also refers to a number of different iterative and incremental software development methodologies that share common principles and practices. These methodologies emphasize people, communication and the ability to adapt to change rather than the process, tools and predictive planning. The methodologies "are processes that support the agile philosophy" [26,32,35] and each of them consists of individual practices and techniques.

Many of the agile methodologies (then called as lightweight) were created in the 1990s [26,36] as an alternative to the traditional sequential (waterfall), document-centric and often heavyweight software development processes and their problems. Although agile methodologies are relatively new, some of their concepts like Iterative and Incremental Development (IID) can be traced back to the 1930s [30,37,38]. NASA has used IID in software projects since the 1960s and IBM from the 1970s [37,39] and it has been promoted by several software development thought leaders since the 1970s [37,38]. Also the ideas of Lean Product Development (used and propagated by Toyota in automobile production) have influenced the development of agile methodologies [35,36] as they spread to North America and to the IT community at large in the 1980s[40]. The actual term agile software development was coined in 2001 when 17 lightweight methodologists got together [25,36] and they wrote the Agile Manifesto based on four values as shown below:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan)

According to Fowler and Highsmith[41] the manifesto is also accompanied by the following 12 principles that reflect its four values as illustrated in Table 1.

Table 1:-*Principles of the Manifesto for Agile Software Development[41]*

Principles of Principles of the Manifesto for Agile Software Development
1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity - the art of maximizing the amount of work not done - is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

In general, it is confirmed that the agile methodologies share many common practices like iterative and incremental development and delivery, adaptive planning and put open face-to-face communication and people before documentation, processes and tools[25, 26] In addition, in addition to working and delivering agile methodologies in short iterations, an agile team acts as one sharing a common purpose [42]. Irrespective of the positions, problems are solved together. As the main means of correspondence, records are no longer transferred from one expert to another.

It sees programming as a detailed art. In addition to writing the code, it also requires the code's technological design (modeling) and checking. Agile teams concentrate company targets by delivering complete user-valued features in customer specified order to optimize the ROI [42].

Teams also have an onsite client representative who works regularly with the team to provide input and identify software specifications [32,38]. This face-to-face direct contact [43] (and other practices) helps the team to build the program without requiring comprehensive written documentation (like a traditional software requirement specification).
The following section will explain in detail about XP. This highlighted in the next section elements related to XP method, for instance the definitions, values and the twelve practices.

EXTREME PROGRAMMING PRACTICES (XP)

XP was created by Kent Beck and Martin Fowler [46] while working for Chrysler Corporation and was first published in his book[47].The name reflects the idea that teams should take good, proven engineering practices to the extreme[36]. XP stresses "customer satisfaction through rapid creation of high-value software, skilful and sustainable software development techniques and flexible response to change" [37]. According to Shore and Warden [32] XP project life-cycle is divided into 1-4 week iterations (preference on the shorter) and the teams are relatively small (5-20 members). In fact, the XP method involves a main four values. In the following paragraph discusses these values in more detail [48,49]

Communication: XP encourages the team members and users to own a shared view on requirements. As a result of continuous communication between the team members as well as with the user, the knowledge about the new system becomes unified. Therefore, there are fewer possibilities of ambiguities and misunderstandings on requirements. Projects developed with XP show that good results can be obtained using sheets of papers to collect user requirements, wall boards to show diagrams and other project-relevant information, and shared workspaces to maximize face-to-face communication.

Feedback: Developers must always provide a means of collecting knowledge about the phase of creation. There are several aspects of feedback: the system, client, and team members. System input and team members seek to provide project leaders with fast indicators of the su cess of the project, while customer feedback provides functional and acceptance checks.

Simplicity: It's one of the ideals that XP directly supports. A simple design often

takes less time than a complicated one to complete.

Thus, XP allows developers to begin with the simplest solution. It is then possible to add extra features later. The simplest thing that might work, programmers do and leave the machine in the simplest state. This boosts the overall development pace while also maintaining a focus on working applications.

Courage: XP helps the members of the team to make choices that help XP practice implementation. In order to refactor the software code, the team members need courage. To promote the introduction of potential improvements, the team members review the current framework and alter it.

Furthermore, bravery can involve eliminating sections of obsolete source code, no matter how much effort has been made to construct these parts.

The focus of XP's practices is on programming and code quality, but it is also a philosophy focused on teamwork and teams. XP does not include other comprehensive work items (like a specifications specification document) but software code and test cases. The recommended way of dealing with specifications and design is oral communication. It is expected that the entire team, including clients, developers and managers, will work together in the same project space to rapidly produce high-quality applications [26,50]

In an XP project, the customer's job is to document software requirements/features as user stories, prioritize these stories by their business value, and write and conduct tests that prove that the stories are implemented as planned.

The function of the XP programmer is flexible and does not differentiate between programmers, designers, testers, etc. All programmers work as a team and share roles that in a non-XP project might be delegated to particular individuals. The programmers are responsible for creating job estimates for the user stories and writing automated unit tests for all they program, in addition to the design and development tasks. The team can also have an XP coach or a project manager who supervises the use of XP procedures and keeps the job running[25]. In addition to the values and principles, XP includes twelve software engineering practices which it combines for greater synergy as shows in Figure 1.

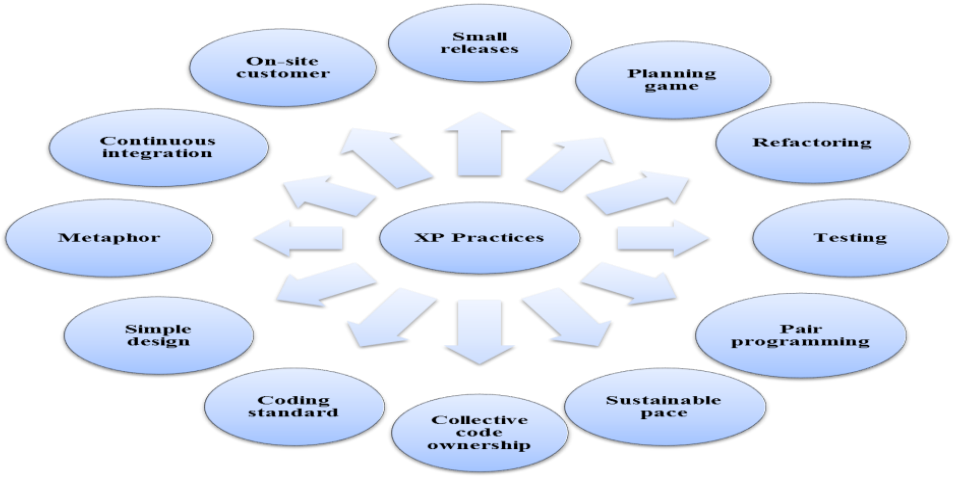

Fig.1:- *Original XP practices [32,34,42,49]*

On site customer: is the practice which deals with the communication aspects among the customers and developmental team. It is an extremely important towards producing quality software. In another words, it concerns about many characteristics in software engineering. For example, the number and type of meetings is a main target for this practice. It has used to collect the software's requirements and the feedback for versions previews of software. Moreover, it refers to how many times that the team spends with the customer to set immediate and continuous feedback when developing software. The customers have to be available full-time for the development team. On site customer practice is looking for explaining how to communicate with customers and get the requirements and feedback from them and how long take every meeting. As well as, the activity of customers in software development [1,2].

Planning game: it refers to agreed statement by the client that demonstrates what the system can do, determine the target functions, and constrains of system. The planning practice deals with writing and documenting methods for system needs and function and how to get the requirements from clients. As well as, estimate the development time and prioritize the software requirements [3,4].

Collective Code Ownership: it considers that the developed code is belonging to the development team rather than the individual member for the software. The code must be available and accessible to all developers of team. For this reason, every developer is going to contribute and add a new idea to all parts of software at anytime and anywhere they gets an opportunity to add new value and feel it is an important without asking for permission. As a final point, this practice makes the code as a one repository and reachable for all the programmer of project team [5,6].

Coding Standard: in software engineering industry, every project has a set of coding rules. The main idea of this practice is that developers should that the entire developers of project team agree to adhere and follow a common set of coding standards on a software project throughout the project. As well, this practice discuss that the type of standard which use in this project and what the responsibility of developers for that selected standard. Just like there is value in following common coding conventions, clean code that follows your chosen coding guidelines is easier to understand and evolve than code that doesn't, there is similar value in following common modeling conventions. In addition, developers also incorporate coding standard practice with note taking technique by adding comments to their code. By applying this coding standard, the code written by different team members is easier to understand and helps software reuse in the future projects [7-9].

Continuous Integration: this practice refers to developers is able to merge code into a shared depository several times a day. It involves in continuous quality control as small pieces of work are tested frequently to provide continuous feedback on the project's progress and to improve the quality of software. Moreover, it cares about how the development team uses it and what the tool of this practice. On other side, it replaces the traditional practice of applying quality control only after completing all development. It helps for reducing developments risks. Continuous integration guarantees that working software is available to employ with new features. It allow developers to learn, interact, and share knowledge to enhance learning process[1, 10].

Frequent Releases: this practice refers to a team could launch code/module to the user frequently and listening to feedback, whether crucial or appreciative. It shortens release cycle to speed the feedback from the client. In condition, the requirements often change, one keeps release cycles short and ensures that each release produces a beneficial software that makes business value for the client. An early version of the project is put into production quickly, small iteration later. In the end of every version, the client reviews the interim product; identify defects and adjusting changes and future requirements to improve the software functions and features [3, 11].

Sustainable Pace (40-Hours week): sometime it is known as 40-weeks hours. Extreme programming teams are in it for the long term. They work hard, and at a pace that can be sustained indefinitely. This means that they work overtime when it is effective, keeping them fresh, healthy, as to reduce as much as possible mistakes and that they normally work in such a way as to maximize productivity week in and week out. On other hand, they do not work for more than 40 hours for week as a rule and never overtime for two consecutive weeks. It is pretty well understood these days that death much quality software. XP teams are in it to win, not to die [12,13].

Pair programming: this practice is one of the primary practices of Extreme Programming (XP). It is means that two programmers can work and writes all production code together as a pair on the single computer, one is the driver (writes code) while the other the observer will assist the driver and suggest a solution. On the other word, one writes the code and, at the same time, another reviews the code for correctness and understandability. They have selected according to specific criteria and they can switch their tasks. It ensures that all written code is reviewed by at least one other developer, resulting in better design, better testing, and better code. It may seem inefficient to have two developers doing "one developer's job", but the reverse is true. Research on pair programming shows that pairing produces better code in about the same time as programmers working singly [14,15].

Test First Programming: this kind of practice is known as unit test and test first design also. It means that the software's programmers make a prior test before beginning the coding process. It helps programmers to really get what needs to be developed. The requirements of software are nailed down firmly by these tests. It clears the understanding a specification written in the form of executable code. It is often very difficult to test some software systems. These systems are typically built code first and testing second, often by a different team entirely. By creating tests first the programming will be influenced by a desire to test everything of value to your customer. The design will reflect this by being easier to test [6,16].

Simple design: XP follows the principle 'keep it simple.' That is, in XP, designs must be easy to implement and a developer should be able to make necessary amendments when required [65,66]

Refactoring: it is the process of improving the design of an artifact without changing its functionality. Refactoring should be done on an ongoing basis throughout development of the artifact. Better arrangements for parts of an artifact can provide, for example, support to other ideas. On the other hand, allowing poorly structured ideas to exist in a project is a risk that accumulates over weeks of development [67].

Metaphor: a metaphor represents a coherent view of the system that makes sense to both the business and technical sides and represents "what we are trying to

do." The metaphor is sometimes embodied in a single user story that portrays this idea and gives everyone the system basics. In a sense, the metaphor serves as the high-level software architecture [68]. At its best, the metaphor is a simple evocative description of how the program works, such as "this program works like a hive of bees, going out for pollen and bringing it back to the hive" as a description for an agent-based information retrieval system[69].

Based on the discussion above and the previous studies, Table 2 distinguishing the XP practices which address the software quality and those which address the development process quality. This mapping highlights the different aspects concerning quality with respect to XP practices.

***Table 2**:-XP practices mapping with respect to quality subjects[70]*

XP practices address the software quality	XP practices address the development process quality	Quality aspect / Influence level
Simple design		High
Testing	Planning game	
Refactoring	Customer on-site	
Continuous integration	Pair programming	
	Collective code ownership	
Small releases	Metaphor	Normal
Coding standard	40- hour week	

The Adoption of Agile Practices

Agile methodologies were developed as a remedy to the failure of predictable manufacturing concepts, such as the waterfall life-cycle, big up-front specifications and speculative planning as they were misapplied to software development. Besides giving flexibility and focusing on delivering customer value, where [25] stated that the agile methodologies reduce the risk of building a wrong product by:

1- Working on the requirements with an on-site customer,

2- Eliciting stakeholder feedback early and often with working software, and

3- Adapting development to changing requirements based on that feedback.

Agile development also reduces the risk of building the right product wrong with test-driven development, continuous integration and other practices and techniques concentrating on software quality. When working software is evaluated and tested in every sprint, requirements and design issues and also software defects are discovered much earlier than in waterfall type projects where testing is done only once at the end of the project. Also, the risk of getting stuck in the requirements or design phase in an unclear project is negated as agile development ensures that actual implementation is done in every sprint[40]. Aguanno also points out two issues related to agile development that needs to be considered. Firstly, a self-organizing, empowered agile team tends to locally optimize their way of working in a particular project, which can cause problems in enterprise project/portfolio management. Secondly, agile methodologies are not formal enough for life-critical systems development as they lack the necessary design reviews and evaluations needed to discover possible safety issues.

Furthermore, agile methods have many

significant attribute one of them is an adaptive development process, which draws on the two lean principles of "amplifying learning" and "decide as late as possible." The lean principle "amplifying learning" is based on the concept that Development is an exercise in discovery while production is an exercise in reducing variation, and for this reason, a lean approach to development results in practices that are quite different than lean production practices." [71]. The lean principle "decide as late as possible" provides a capacity for change by delaying decisions as late as possible. ASDMs follow with these principles by emphasizing adaptive software development, which requires iterative and incremental development through productive feedback. Satzinger, Jackson, and Burd [72] mentioned that some projects were reasonably predictable and could be managed sequentially but most projects are less predictable, demanding an iterative and adaptive approach to development.

Small-Medium-Large Scale Project

Most agile methods have primarily been applied to small to medium size projects such as internet and web-based information systems. It is not clear if agile methods are used on large-scale projects that they can provide end-users with the desired quality in a timely manner [73]. However, some researchers have reported that large-scale and complex projects have benefited from suitably tailored agile development methods [74-77].

As well as, Bowers et al [74] examined whether the XP method can handle large-scale and life-critical software systems. The authors adopted the XP method to redesign their public safety communication systems, which consists of over a million lines of C language code. They indicated that a suitably adapted agile development process (in particular XP) was ideal for long-term projects and the development of large systems. This is contradictory to the preferences of many information technology (IT) managers who often consider XP as a slightly chaotic methodology. Lippert et al [75] mentioned that they followed the recommended practice of adapting XP to their specific project. They also developed methodological extensions to XP for use in a number of areas in which questions and problems frequently occur. The majority of studies on large-scale projects have been conducted using the XP method, which was initially designed for small-scale projects with less than 10 developers and a product that would not be excessively complex[80].

There studies used the XP method to mitigate risks with early, frequent feedback. However, they did not use every part of the XP method. Instead, they adopted some practices, dropped others and supplemented others with practices from other fields. This paper revealed the possibilities for applying the XP method to large-scale and life-critical projects if the XP method was modified to fit into the specific application development environment. Lippert et al[75] also examined whether the XP method was appropriate for large and long term projects.

In dead, each agile method is a unique system or software development methodology according to the definition of Avison and Fitzgerald[78] , each agile method has a different purpose. For example, XP is specifically designed for software development in high change environments, for satisfying customer needs, and for maintaining effective teams [80]. Scrum focuses on project management of iterative development[79], and Adaptive System Development (ASD) is a framework for managing software projects under intense time pressure [41].

Strengths and Weaknesses of XP Method

Many researchers indicate the strengths and weaknesses of XP method. Table 3 depicted these cons and pros of the XP based on number of the researchers.

***Table 3**:-Summary of the Common Strengths and Weaknesses of XP*

Strengths of XP Method	
XP method helps the software industry for shorter release of functional software, where the customers are always contacted to ask for the highest priority features in the software.	Beck, 2000; Fruhling & Vreede, 2006; Xu, 2009 [80-82]
XP method saves the project against the cancellation with the help of periodic releases.	Beck, 2000; Guha et al., 2011.[80,83]
XP method always focuses on the highest priority tasks; therefore false features are not prioritized during the development of the software, as it gives the freedom to the developers and testers to give their feedbacks upon the release time and cost of the software which will helpful for interaction with the clients via the business people.	Beck, 2000; Munassar & Govardhan, 2010; Xu, 2009.[80,82-84]
XP method is more flexible and includes more explicitly the needs and intentions of all project participants.	Beck, 2000; Fruhling & Vreede, 2006; Xu, 2009.[80-82]
By test driven development practices, XP method resulting in less errors and acceptance of changing requirements.	Beck, 2000; Fruhling & Vreede, 2006; Munassar & Govardhan, 2010.[80,81,84]
XP method is suited for single project, developed and maintained by a single team. It cannot be implemented in the system where developers don't work well with each other and like to work on their own.	Beck, 2000; Guha et al., 2011; Hneif & Hock Ow, 2009.[80,83,85]
Weaknesses of XP Method XP method is not suitable for medium and large scale projects.	Munassar & Govardhan, 2010; Mushtaq & Qureshi, 2012; Hneif & Hock Ow, 2009.[7,84,85]
XP method is not suitable to be implemented in an environment where a customer or manager insists on a complete specification or design before they begin programming.	Beck, 2000, Turk et al., 2002; Xu, 2009.[80, 33,82]
Lack of project management practices.	Beck, 2000; Turk et al., 2002; Mushtaq, 2012[7,80,33]
Lack of documentation though the development lifecycle.	Qureshi, 2011; Munassar & Govardhan, 2010; Guha et al., 2011; Paulk, 2001. [83,84,86]
Developers must be experienced.	Paulk, 2001; Munassar & Govardhan, 2010[84,86].

RELATED WORKS

The research community has devoted a great deal of attention to agile software development since the agile manifesto was created in 2001. Dingsøyr, Nerur, Balijepally and Moe [87] referred that there are around 32 articles from 2003 until 2011 addressed the agile software development and their applying such methods in industry. Moreover, the XP was described as the most common agile methods. These articles were focused on understanding of agile concepts, adoption and/or adaptation of agile, and evaluation of adoption issues in environments that are not inherently conducive to agile. The

reviewing of the previous studies would illustrated the applied of Extreme programming methodologies in different area as showed in Table 4.

Table 4:-*Summary of the Application Extreme programming Practices*

Authors	Year	Type of the study	Finding
Sfetsos, Angelis & Stamelos[88]	2006	Mix methods	• The results have shown that companies, facing various problems with common code ownership, on-site customer, 40-hour week and metaphor, prefer to develop their own tailored XP method and way of working-practices that met their requirements. • Pair programming and test-driven development were found to be the most significant success factors.
Salo & Abrahamsson[58]	2008	Quantitative	• The outcomes of study showed that the organizations are able to apply the two agile methods, namely, XP and Scrum, and their individual practices in their projects and report fairly positive results of their application; and the most used XP practices among the respondents. • Moreover, the experienced usefulness of the practices was clearly higher than the expected usefulness among the respondents not having applied the practices of XP and Scrum in their projects.
Omar, Syed-Abdullah, &Yasin, A. [89]	2010	Qualitative	• The output shows that the adopting agile-XP practices have been successfully implemented in this centre; despite the XP practices have not fully adopted. This is because organization culture may affected the adoption.
Haider & Ali[90]	2011	Mix methods	• The outcome of this study shows that the using of Pair programming as an effective software development technique as well as a pedagogical tool. Furthermore, the use of pair programming also effects performance in distributed software development, and positively impacts the social practices (human or social factors).
Ghani, Izzaty, & Firdaus[91]	2013	Qualitative	• The results indicated that software development using XP method delivered quickly.
Mohamed, Farvin, Baharom, & Deraman,[92]	2014	Quantitative	• All of the respondents agreed that agility should be considered during software development in order to produce high quality software. • Software practitioners in Malaysia are gradually implementing agile based software development; but there still exist among them who have never heard about it. • The most implemented agile methods are XP and Scrum.
Omar & Abdullah[93]	2015	Quantitative	• The findings showed that the use of agile methodology does not significantly affect work-related well-being. • Agile practices, such as pair programming, continuous integration, and frequent release, are able to induce teams to work closely and experience higher well-being.

CONCLUSION

The aim of this article is to identify the best practices for evaluating the quality of Extreme programming (XP) implementation. XP has been chosen in this article because it is one of the most prevalent software development methodologies. The current study identified twelve practices based on the previous studies for evaluating the XP quality implementation. Many researchers state that these practices must be used together and support each other to get high quality XP implementation.

REFERENCES

1. Wu, B. H. (2011, March). On software engineering and software methodologies a software developer's perspective. In *International Conference on Information Science and Technology* (pp. 155-162). IEEE.
2. Senapathi, M., & Srinivasan, A. (2012). Understanding post-adoptive agile usage: An exploratory cross-case analysis. *Journal of Systems and Software*, *85*(6), 1255-1268.
3. Päivärinta, T., & Smolander, K. (2015). Theorizing about software development practices. *Science of Computer Programming*, *101*, 124-135.
4. Burman, E. (2015). Agile in action: Hybrid methodologies in practice.
5. Hass, K. B. (2007). The blending of traditional and agile project management. *PM world today*, 9(5), 1-8.
6. Abrahamsson, P., Conboy, K., & Wang, X. (2009). ―Lots done, more to do‖: the current state of agile systems development research.
7. Mushtaq, Z., & Qureshi, M. R. J. (2012). Novel Hybrid Model: Integrating Scrum and XP. *International Journal of Information Technology and Computer Science (IJITCS)*, *4*(6), 39.
8. Kalermo, J., & Rissanen, J. (2002). Agile software development in theory and practice. University of Jyväskylä.
9. Chandra Misra, S., Kumar, V., & Kumar, U. (2010). Identifying some critical changes required in adopting agile practices in traditional software development projects. *International Journal of Quality & Reliability Management*, *27*(4), 451-474.
10. Blokdijk, A. (2014). Planning and design of information systems: Academic Press.
11. Pressman, R. S. (2005). Software engineering: a practitioner's approach: Palgrave Macmillan.
12. Chung, L., Nixon, B. A., Yu, E., & Mylopoulos, J. (2012). Non-functional requirements in software engineering. *Springer Science & Business Media*
13. Pressman, R. S., & David Brian, L. (2009). Web engineering:: a practitioner's approach.
14. Pickering, C. (2001). Building an Effective E-project Team. E-Project Management Advisory Service, *Cutter Consortium*, *2*(1).
15. McConnell, S. (2004). Code complete: Pearson Education.
16. Miller, J. H., & Page, S. E. (2009). Complex adaptive systems: an introduction to computational models of social life: an introduction to computational models of social life: Princeton university press.
17. Cyganek, B., & Siebert, J. P. (2011). An introduction to 3D computer vision techniques and algorithms: John Wiley & Sons.
18. Boehm, B. (2006). A view of 20th and 21st century software engineering. Paper presented at the Proceedings of the 28th international conference on Software engineering.
19. Unterkalmsteiner, M., Gorschek, T., Cheng, C. K., Permadi, R. B., & Feldt, R. (2012). Evaluation and measurement of software process improvement—a systematic literature

review. *Software Engineering, IEEE Transactions on, 38*(2), 398-424
20. Highsmith, J. (2013). Adaptive software development: a collaborative approach to managing complex systems: Addison-Wesley.
21. Santos, R. P. d. (2014). ReuseSEEM: an approach to support the definition, modeling, and analysis of software ecosystems. *Paper presented at the Companion Proceedings of the 36th International Conference on Software Engineering.*
22. Valacich, J., George, J., & Hoffer, J. (2009). Essentials of system analysis and design: Prentice Hall Press.
23. Bird, M. (2007). Comprehensive Examination Written Responses Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy. Capella University
24. Douglas, I. (2006). Issues in software engineering of relevance to instructional design. *TechTrends, 50*(5), 28-35
25. Leffingwell, D. (2010). Agile software requirements: lean requirements practices for teams, programs, and the enterprise: Addison-Wesley Professional.
26. Stober, T., & Hansmann, U. (2010). Best Practices for Large Software Development Projects: *Springer.*
27. Cagle West, M. (2010). Cagle ProQuest LLC.
28. Bustard, D., Wilkie, G., & Greer, D. (2013). The maturation of agile software development presented at the Engineering of Computer Based Systems (ECBS), 2013 20th IEEE International Conference and Workshops on the. Principles and practice: observations on successive industrial studies in 2010 and 2012. Paper
29. Leau, Y. B., Loo, W. K., Tham, W. Y., & Tan, S. F. (2012). Software development life cycle AGILE vs traditional approaches. Paper presented at the International Conference on Information and Network Technology.
30. Eckstein, J. (2013). Agile software development in the large: Diving into the deep: Pearson Education.
31. Cano, S. P., González, C. S., Collazos, C. A., Arteaga, J. M., & Zapata, S. (2015). Agile Software Development Process Applied to the Serious Games Development for Children from 7 to 10 Years Old. *International Journal of Information Technologies and Systems Approach (IJITSA), 8*(2), 64-79.
32. Shore, J., & Warden, S. (2008). The Art of Agile Development O'Reilly Media Inc: Shroff Publishers and Distributors Pvt. Ltd.
33. Turk, D., France, R., & Rumpe, B. (2014). Assumptions underlying agile software development processes. arXiv preprint arXiv:1409.6610.
34. Turk, D., France, R., & Rumpe, B. (2014). Limitations of agile software processes. arXiv preprint arXiv:1409.6600.
35. Soundararajan, S., Arthur, J. D., & Balci, O. (2012). A methodology for assessing agile software development methods. *Paper presented at the Agile Conference (AGILE),* 2012.
36. Sliger, M., & Broderick, S. (2008). The software project manager's bridge to agility: Addison-Wesley Professional.
37. Larman, C. (2004). Agile and iterative development: a manager's guide: Addison-Wesley Professional
38. Petersen, K., & Wohlin, C. (2009). A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. *Journal of Systems and Software, 82*(9), 1479-1490.
39. Kruchten, P. (2013). Contextualizing agile software development. *Journal of Software: Evolution and Process, 25*(4), 351-361.

40. Aguanno, K. (2004). 101 Ways to Reward Team Members for $20 (or Less!): Multi-Media Publications Inc.
41. Highsmith, J. (2000). Retiring Lifecycle Dinosaurs A look at Adaptive Software Development, an alternative to traditional, process-centric software management methods. *Software testing and quality engineering*, *2*, 22-30.
42. Cohn, M. (2005). Agile estimating and planning: Pearson Education.
43. Cockburn, A., 2007. Agile Software Development: A Cooperative Game. 2nd Edn., Addison Wesley, ISBN: 0-321-48275-1, pp: 504.
44. Elssamadisy, A. (2008). Agile adoption patterns: a roadmap to organizational success: Addison-Wesley Professional.
45. Alite, B., & Spasibenko, N. (2008). Project Suitability for Agile methodologies. Umeå School of Business
46. Wells, D. (2009). Agile process. extreme programming: a gentle introduction.
47. Beck, K. (1999). Embracing change with extreme programming. Computer, 32(10), 70-77.
48. Rittenbruch, M., McEwan, G., Ward, N., Mansfield, T., & Bartenstein, D. (2002). Extreme participation-moving extreme programming towards participatory design. Paper presented at the PDC2002 Proceedings
49. Darwish, N. R. (2013). Towards an Approach for Evaluating the Implementation of eXtreme Programming Practices. *International Journal of Intelligent Computing and Information Sciences (IJICIS),* Ain Shams University, 13(3).
50. Martin, R. C. (2003). Agile software development: principles, patterns, and practices: Prentice Hall PTR.
51. Syed-Abdullah, S. L., Omar, M., Hamid, M. N. A., bt Ismail, C. L., & Jusoff, K. (2009). Positive affects inducer on software quality. *Computer and Information Science, 2*(3), p64.
52. Wood, S., Michaelides, G., & Thomson, C. (2013). Successful extreme programming: Fidelity to the methodology or good teamworking? *Information and Software Technology, 55*(4), 660-672.
53. Jun, L., Qiuzhen, W., & Lin, G. (2010). Application of agile requirement engineering in modest-sized information systems development. Paper presented at the Software Engineering (WCSE), 2010 Second World Congress on.
54. Abrantes, J. F., & Travassos, G. H. (2011). Common agile practices in software processes. Paper presented at the Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on.
55. Lindstrom, L., & Jeffries, R. (2004). Extreme programming and agile software development methodologies. *Information systems management, 21*(3), 41-52.
56. Aveling, B. (2004). XP lite considered harmful? Extreme Programming and Agile Processes in Software Engineering (94-103): Springer.
57. Omar, M., Abdullah, S., & Lailee, S. (2013). Agile practices: A cognitive learning perspective.
58. Salo, O., & Abrahamsson, P. (2008). Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of Extreme Programming and Scrum. *Software, IET, 2*(1), 58-64.
59. Sison, R., & Yang, T. (2007). Use of Agile Methods and Practices in the Philippines. Paper presented at the Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific.
60. Kongyai, B., & Edi, E. (2011). Adaptation of Agile Practices: A Systematic Review and Survey

61. Hummel, M. (2014). State-of-the-Art: A Systematic Literature Review on Agile Information Systems Development. Paper presented at the System Sciences (HICSS), 2014 47th Hawaii International Conference on.
62. Begel, A., & Nagappan, N. (2008). Pair programming: what's in it for me? Paper presented at the Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement
63. Rumpe, B., & Schröder, A. (2014). Quantitative survey on extreme programming projects. arXiv preprint arXiv:1409.6599.
64. Lemos, O. A. L., Ferrari, F. C., Silveira, F. F., & Garcia, A. (2012). Development of auxiliary functions: Should you be agile? an empirical assessment of pair programming and test-first programming. Paper presented at the Proceedings of the 34th International Conference on Software Engineering.
65. Harrison, N. B. (2003). A study of extreme programming in a large company. Avaya Labs.
66. Singhal, A., & Banati, H. (2014). FISA-XP: an agile-based integration of security activities with extreme programming. *ACM SIGSOFT Software Engineering Notes*, *39*(3), 1-14.
67. Siebra, C., Mozart Filho, S., Silva, F. Q., & Santos, A. L. (2008). Deciphering extreme programming practices for innovation process management. Paper presented at the Management of Innovation and Technology, 2008. ICMIT 2008. 4th IEEE International Conference on
68. Maurer, F., & Martel, S. (2002). Extreme programming: Rapid development for Web-based applications. *IEEE Internet computing*(1), 86-90.
69. Jeffries, R. (2003). Extreme Programming and Agile Software Development Methodologies: CRC Press LLC.
70. Dubinsky, Y., & Hazzan, O. (2002). Improvement of software quality: Introducing extreme programming into a project-based course. Paper presented at the 14th International Conference of the Israel Society for Quality.
71. Poppendieck, M., & Poppendieck, T. (2003). Lean software development: an agile toolkit: Addison-Wesley Professional.
72. Satzinger, J. W., Jackson, R. B., & Burd, S. D. (2005). Object-oriented Analysis and Design: With the Unified Process: Thomson Course Technology
73. Marrington, A., Hogan, J. M., & Thomas, R. (2005). Quality assurance in a student-based agile software engineering process. Paper presented at the Software Engineering Conference, 2005. Proceedings. 2005 Australian
74. Bowers, J., May, J., Melander, E., Baarman, M., & Ayoob, A. (2002). Tailoring XP for large system mission critical software development Extreme Programming and Agile Methods—XP/Agile Universe 2002 (pp. 100-111): Springer.
75. Lippert, M., Becker-Pechau, P., Breitling, H., Roock, S., Schmolitzky, A., Wolf, H., & Heinz, Z. (2003). *Developing complex projects using XP with extensions. Computer(6)*, 67-73
76. Cao, L., Mohan, K., Xu, P., & Ramesh, B. (2004). How extreme does extreme programming have to be? Adapting XP practices to large-scale projects. Paper presented at the System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on.
77. Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, D., Kähkönen, T. (2004). Agile

software development in large organizations. *Computer, 37*(12), 26-34.
78. Avison, D., Cole, M., & Fitzgerald, G. (2006). Reflections on teaching information systems analysis and design: from then to now. *Journal of Information Systems Education, 17*(3), 253.
79. Schwaber, K., & Beedle, M. (2002). gilè Software Development with Scrum.
80. Beck, K. (2000). Extreme programming explained: embrace change: Addison-Wesley Professional
81. Fruhling, A., & Vreede, G.-J. D. (2006). Field experiences with eXtreme programming: developing an emergency response system. *Journal of Management Information Systems, 22*(4), 39-68.
82. Xu, B. (2009). Towards high quality software development with extreme programming methodology: practices from real software projects. Paper presented at the Management and Service Science, 2009. MASS'09. International Conference on.
83. Guha, P., Shah, K., Shukla, S. S. P., & Singh, S. (2011). Incorporating Agile with MDA Case Study: Online Polling System. arXiv preprint arXiv:1110.6879.
84. Munassar, N. M. A., & Govardhan, A. (2010). A comparison between five models of software engineering. IJCSI, 5, 95-101
85. Hneif, M., & Hock Ow, S. (2009). Review of Agile Methodologies in Software Development. *International Journal of Research and Reviews in Applied Sciences, 1*(1). 1-8.
86. Paulk, M. (2001). Extreme Programming from a CMM Perspective. *IEEE Software, 18*(6), 19-26.
87. Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software, 85*(6), 1213-1221.
88. Sfetsos, P., Angelis, L., & Stamelos, I. (2006). Investigating the extreme programming system–An empirical study. *Empirical Software Engineering,11*(2), 269-301.
89. Omar, M., Syed-Abdullah, S.-L., & Yasin, A. (2010). Adopting Agile Approach: A Case in Malaysia
90. Haider, M. T., & Ali, I. (2011). Evaluation of the Effects of Pair Programming on Performance and Social Practices in Distributed Software Development.
91. Ghani, I., Izzaty, N., & Firdaus, A. (2013). Role-based Extreme Programming (XP) for secure software development. *Science International (Lahore), 25*(4 (Spe), 1071-1074
92. Mohamed, P., Farvin, S., Baharom, F., & Deraman, A. (2014). An Exploratory Study on Agile based Software Development Practices. *International Journal of Security & Its Applications, 8*(5).
93. Omar, M., & Abdullah, S. L. S. (2015). The Impact of Agile Methodology on Software Team's Work-Related Well-Being. *International Journal of Software Engineering & Its Applications, 9*(3).