THE INVESTIGATION ON THE BEST PRACTICES OF EXTREME PROGRAMMING (XP) QUALITY IMPLEMENTATION AT UUM IT

RANA ALAULDEEN ABDULRAHMAN

MASTER OF SCIENCE (INFORMATION TECHNOLOGY)
UNIVERSITI UTARA MALAYSIA
2015

Permission to Use

In presenting this dissertation in partial fulfilment of the requirements for a postgraduate degree from Universiti Utara Malaysia, I agree that the Universiti Library may make it freely available for inspection. I further agree that permission for the copying of this dissertation in any manner, in whole or in part, for scholarly purpose may be granted by my supervisor(s) or, in their absence, by the Dean of Awang Had Salleh Graduate School of Arts and Sciences. It is understood that any copying or publication or use of this dissertation or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to Universiti Utara Malaysia for any scholarly use which may be made of any material from my dissertation.

Requests for permission to copy or to make other use of materials in this project dissertation, in whole or in part, should be addressed to:

Dean of Awang Had Salleh Graduate School of Arts and Sciences
UUMCollege of Arts and Sciences
Universiti Utara Malaysia
06010 UUM Sintok

Abstrak

Kejuruteraan perisian (SE) memainkan peranan yang penting dalam meningkatkan kesejahteraan masyarakat melalui penggunaan perisian yang berkualiti tinggi. Kebanyakan projek perisian gagal disebabkan organisasi perisian tidak mempraktis amalan pembangunan perisian yang sewajarnya. Sehubungan itu, organisasi perisian perlu mempunyai metodologi pembangunan perisian yang baik bagi memenuhi keperluan pihak pemegang taruh. Salah satu metodologi pembangunan perisian dalam SE yang semakin berkembang penggunaannya adalah metodologi *Extreme Programming* (XP). Metodologi ini merupakan pendekatan baru dalam SE yang mampu meningkatkan kualiti perisian dan berupaya mengurangkan masa pembangunan perisian dan kos. Walau bagaimanapun, tahap penggunaan metodologi ini di kalangan pembangun perisian di Pusat UUM IT masih tidak jelas. Oleh yang demikian, kajian ini bertujuan untuk mengkaji penggunaan amalan XP di pusat ini. UUM IT telah dipilih sebagai kajian kes kerana peranan organisasi ini telah berubah bagi memenuhi permintaan yang tinggi di kalangan masyarakat kampus. Oleh itu, penyelidikan yang memfokuskan kepada kepada 12 amalan XP di UUM IT amat diperlukan. Kajian ini dijalankan dengan menemubual secara separa berstruktur dengan lima (5) pakar dari UUM IT bagi mengenal pasti kejayaan pelaksanaan amalan XP. Hasil kajian telah menunjukkan bahawa sebahagian besar daripada amalan XP digunakan oleh pembangun perisian di UUM IT tetapi perlu dipertingkatkan Sebaliknya, beberapa amalan seperti *pair programming* dan *test first programming* tidak digunakan oleh pembangun perisian di UUM IT. Ini disebabkan jenis dan sifat projek perisian yang terlibat, dan juga disebabkan oleh personaliti, pengalaman dan tahap pendidikan yang berbeza di kalangan pembangun perisian. Kajian ini menyediakan bukti kualitatif yang dapat membantu pengurus projek perisian dalam membimbing mereka meningkatkan amalan pembangunan perisian bagi menghasilkan perisian yang berkualiti tinggi.

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. This methodology is an emerging SE approach, which is able to increase software quality and hence reducing software development time and cost. However, the level of application of this methodology among software developers in UUM IT centre is still unclear. Therefore, this study aims to investigate the application of XP practices in this centre. UUM IT was chosen as a case study because the role of this organization has changed to meet high demand among campus communities. Thus, research that focuses on the 12 XP practices of UUM IT is highly needed. This study was conducted using a semi–structured interview with five (5) experts from the UUM IT, to identify the successful implementation of the XP practices. The findings have shown that, most of the practices are used by UUM IT developers but need to improve. In contrast, some of the practices such as pair programming and test first programming are not used by the UUM IT developers. This is due to the nature and type of software projects involved, also because of the personality, experiences and the education level differences among developers. This study provides qualitative evident that can assist software project managers to guide them in improving software development practices for producing high quality software.

Acknowledgement

In the name of Allah Gracious and most Merciful. In the first place, I wish to express my deepest gratitude to Allah for providing me with the substance, time, health, strength and patience to participate in this journey to acquire knowledge.
In accomplishing this research, I would like to express my gratitude to Dr. Mazni Omar for supervision, advice, and guidance of this research as well as giving me from her experiences.

My deepest thanks to my father Alauldeen Abdulrahman who put the fundamental of my learning character since I was a child and my mother Luma Alauldeen who sincerely raised me with her caring and gently love. Thank you for your love, your support, your prayers, for everything you did for me during my study.

Table of Contents

List of Tables

List of Figures

List of Appendices

CHAPTER ONE
INTRODUCTION

1.1 Overview

This initial chapter introduces the background on the phenomenon under study, problem statement, research questions, and research objectives. The research scope and significance of this research are also discussed. The chapter ends with the outline of the thesis structure and summary of the current chapter.

1.2 Background of Study

Software engineering (SE) is a domain that deals with engineering discipline in software construction. It has been kept formal and has practical methodologies as guidance in software development. It has been manifested by software life cycle that is composed of requirement elicitation and analysis, design specification, implementation, verification and validation, deployment and maintenance (Wu, 2011). Software development processes are an important part of software engineering, which influence the product outcome (Senapathi & Srinivasan, 2012; Päivärinta & Smolander, 2015). Several studies noted that software projects are considered a failure for many reasons. Tan (2011) refers that the research conducted by Gartner where data was collected from 845 project sample has shown that 42.5% did not deliver all the benefits, 44% were delivered over budget and 42% were not delivered on time. Furthermore, Gulla (2011) mentions that missing methodology is one of the reasons for software failure. The reason of software failure has also been discussed by Haughey (2011), who claims that poor or missing methodologies and tools are among the reasons.

Generally, there are many traditional methodologies for software engineering, such as waterfall, prototyping, iterative and incremental, and spiral. The technique of using a methodology has appeared since the late 1960s and has traditionally been administrated through the waterfall methodology (Hass, 2007). The traditional methodologies connected the empirical and theoretical issues. Hence, they have a lot of troubles in software engineering industry. With regard to these problems, the researchers have shown that the weaknesses of traditional methods come from the lack of theoretical and empirical connection (Abrahamsson, Conboy, & Wang, 2009; Ahlemann, El Arbi, Kaiser & Heck, 2013).

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 (Burman, 2015; Hass, 2007). 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 (Abrahamsson *et al*., 2009). 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 (Mushtaq & Qureshi, 2012).

The key spirit of agile is that software should be completely integrated and tested before the end of iteration. The agile models offer rapidly changing requirements to develop software for the teams. In Malaysia, the current software development practices are focusing on agile-based software development, which reveals that agile-based software development practices are important in order to produce high quality software (Mohamed, Farvin, Baharom & Deraman, 2014). Agile methods are an established process for developing software nowadays (Asnawi, Gravell & Wills 2012; Asnawi, Gravell & Wills, 2014). There is, however, less evidence on their usage among software practitioners in Malaysia (Asnawi *et al*., 2014; Mohamed *et at*., 2014). While the methods have become mainstream in other regions, that is not the case in this country. More specifically, Asnawi *et al* (2012) assert that agile methods are still emerging methodologies in Malaysia, where the adopters are still at a minimum number.

Extreme programming XP is one of the most widespread and most useful methods of agile in software engineering (Conboy & Fitzgerald, 2010; Tessem, 2003). It is a collection of well-known practices in software engineering. It aims to enable successful software development despite ambiguity or constant changing of the requirements. The novelty of XP is based on the way the individual practices are collected and lined up to function with each other. Darwish (2011) states that the main advantage of XP method is the resilience it provides, allowing for easy incorporation of changes. In the same context, according to Darwish (2013), the life cycle of XP methodology has six phases to develop a software, which are exploration, planning, iteration to release, product ionizing, maintenance, and death.

XP practices are suitable for large-scale, complex software development. Furthermore, XP has many characteristics, such as short iterations with small releases and rapid feedback, close customer participation, constant communication and coordination, continuous refactoring, continuous integration and testing, collective code ownership, and pair programming (Mohammed & Rauf, 2015; Mushtaq & Qureshi, 2012; Rumpe & Schröder, 2014).

XP is a software development discipline in the family of agile methodologies that contributes towards quality improvement using a dozen practices. According to beck (2000) and Haider and Ali (2011), XP consists of twelve practices, which are planning game, small releases, metaphor, simple design, testing, refactoring, pair programming, collective code ownership, continuous integration, 40-hour week, on-site customer, and coding standard. Therefore, XP requires direct communication among all members to give the developers a shared view of the system which matches the view held by the users of the system. Thus, most of the previous studies were focused only on the pair programming practice or another practice separately (Agarwal & Deep, 2014; Sillitti, Succi, & Vlasenko, 2012; da Silva Estácio & Prikladnicki, 2015). Moreover, many scholars emphasize the importance of using every practice, such as Beck (2000) who says that "Any one practice does not stand well on its own. They require other practices to keep them in balance". This claim is also supported by Alshehri (2014) and Stellman and Greene (2014).

In addition, the XP approach is very important for software development as mentioned in the literature. Few empirical studies, particularly qualitative approach conducted in Universiti Utara Malaysia, are related to this approach. Based on these

arguments and many other arguments discussed in the next sections, this study exploits the experts' viewpoint to explore the main XP practices that are harnessed in the UUM IT center. In general, in the next section, the researcher will determine the problem statement based on the previous studies related to this phenomenon.

1.3 Problem Statement

Among the agile methodologies, the Extreme Programming (XP) is the one that has received the most attention (Mannaro, Melis & Marchesi, 2004; Rumpe & Schröder, 2014; Syed-Abdullah, Holcombe & Gheorge, 2006; Solinski & Petersen, 2014). It is the most prevalent in agile software development methodology (Cockburn, 2007; Omar, Syed-Abdullah & Yasin, 2011; Salo & Abrahamsson, 2008). In the same context, da Silva Estácio and Prikladnicki (2014) and Al-tarawneh, Abdullah and Ali (2012) and Hummel (2014) state that Extreme Programming (XP) is one of the agile methods most adopted in the industry. Moreover, Omar and Abdullah (2015) argue that by applying XP methodology, software development teams experience higher enthusiasm levels in the most dynamic project. This is also supported by Sison and Yang (2007), who refer that XP increases good relationship among developers. However, even though in XP it is emphasized that the better results are obtained when applying all the practices, it is not clear what the impact on productivity or quality would be if some practices were ignored (Marchesi, 2005). Moreover, it has been found that there are few empirical studies in this research field.

Furthermore, XP method is used for business where time is important and when requirements are not known earlier (Kumar Srivastava, Singh Chanhan & Singh,

2011). However, many organizations remain skeptical regarding XP's value. In the same context, Mohamed, *et al* (2014) declare that only few studies were conducted among Malaysian software practitioners regarding agile practices, whereas most of the studies were performed in the Western countries. On top of that, Omar, Syed-Abdullah and Yasin (2010) refer that a formal approach of developing software amongst software developers in UUM IT is still unclear.

Moreover, XP proposes twelve software development practices to increase productivity and maintain quality (Abdullah, Al-Tarawnehb & Alia, 2012). However, most of the previous studies focus on pair programming (such as, Sillitti, Succi & Vlasenko, 2012 and Rejab, Omar, Mohd and Ahmed, 2011), while a few studies that concentrate on other XP practices on software development have been found (such as, Kuppuswami, Vivekanandan, Ramaswamy & Rodrigues, 2003). This is also confirmed by Hummel (2014), who points out that most empirical research is concerned with the XP practice pair programming, while other practices are neglected. However, XP practices can create a coherent method when they work together (Paulk, 2001). Moreover, most of the previous studies were focused on pair programming in the education sector (Brereton, Turner & Kaur, 2009). Sillitti *et al*. (2012) refer that pair programming can foster knowledge sharing among students. In addition, Canfora, Cimitile and Visaggio (2003) state that pair programming has been gaining acceptance among practitioners and the software development community. This successfully leads to a wide use of pair programming in educational setting as a computer science or software engineering pedagogical tool

especially in programming courses (Cliburn, 2003; Rejab, Omar, Mohd & Ahmed, 2011)

Therefore, because of the many advantages of XP method, the progressive usage of XP in software organizations, and the importance of XP practices for the success of applying this method, there is a need to ensure the proper implementation of XP practices in UUM IT. Consequently, this study focuses on evaluating the degree to which various XP practices are implemented in this center. To achieve this objective, the researcher uses qualitative study to discover the XP practices mostly used in this center and at the same time highlight the poor practices that need to be well-organized or improved.

1.4 Research Questions

Based on the arguments discussed in section 1.3, this study attempts to answer the following questions:

i. What are the best practices of Extreme Programming (XP)?
ii. What is the conceptual model for evaluating the best practices of Extreme programming (XP) quality implementation?
iii. How is the proposed conceptual model validated?

1.5 Research Objectives

The main aim of this study is to investigate the Extreme programming practices used among UUM IT's developers. In order to achieve this aim, the following objectives have been formulated:

i. To identify the best practices of Extreme programming (XP).

ii. To propose the conceptual model for evaluating the best practices of Extreme Programming (XP) quality implementation.

iii. To validate the proposed conceptual model using a case study at UUM IT.

1.6 Research Scope

In this study, the agile methodology is focused on Extreme Programming (XP) methodology, which is one of the most prevalent agile methodologies. Meanwhile, it is one of the agile methods most adopted in the industry. Therefore, the prior literature states that the usage of XP can help improve quality and productivity. More specifically, the current study concentrates on identifying the best practices for evaluating XP software development among software developers in UUM IT. In fact, UUM IT was established in 1989. In the 25 years of its operation, the role of UUM IT has changed to delivering more impact software projects to cater high demand from the campus communities. Therefore, applying good software development practices amongst UUM IT software developers is crucial because it can help the organization to produce better software.

In addition, qualitative approach, semi-structured interviews are used to highlight these practices that may be harnessed in this center. These interviews were conducted with experts who have more than ten years' experience in software development in University Utara Malaysia

1.7 Significance of the Study

This study significantly provides qualitative evidence on the implementation of Extreme Programming practices used by the software developers in UUM IT. It is able to demonstrate whether to what extent the XP practices have been applied in UUM IT. Also, this qualitative evidence can assist the top management in making informed decisions on how to improve the software development practices in UUM IT. This is because applying good software development practices can improve the software delivered to the customers.

In addition, despite the many benefits agile methods can deliver, to date, little work has been published regarding its current usage in developing countries like Malaysia. Furthermore, this kind of study is also lacking in the Southeast Asia region. Therefore, this research strives to enrich the literature by identifying the key XP practices (as a kind of the agile methodologies) used in Malaysia in general and UUM IT in particular.

1.8 Organization of the Dissertation

i. Chapter one: Overview

   This chapter discusses the issues related to the phenomenon. In addition, it illustrates the objectives of the study, the scope of the study, and the significance of the current study.

ii. Chapter two: Review of Literature

   Most of the related studies are discussed in this chapter. In addition, the concepts and the agile methodologies as well as the practices are discussed.

iii. Chapter three: Research Methodology

The research process that assists to achieve the main objectives of the present research is highlighted. The qualitative approach is discussed as the technique for collecting data from the participants through semi-structured interview.

iv. Chapter four: Discussion of Results and Findings

The preparation and analysis of interviews carried out with experts by using manual methods and using tools is discussed.

v. Chapter five: Conclusion

This chapter discusses the objectives of study. In addition, it highlights the limitations and future work directions.

1.9 Summary of Chapter One

Software plays an important role in the modern world. In addition, the development of software has always been regarded as a difficult task. Thus, the current study aims to explore how to use the most permanent agile methodologies called Extreme Programming (XP) in the computer centers within public universities in Malaysia. Based on the former studies, there is a lack of attention towards the use of agile methodologies in the Southeast Asia region, and especially in Malaysia. Literature also shows that little work has been published regarding XP practices. This chapter also presents the research questions, the research significance, and the scope of the study.

CHAPTER TWO
REVIEW OF LITERATURE

2.1 Introduction

Through this chapter, a review of literature related to agile software development and Extreme programming practices methodology will be explained. Section 2.2 will be discussed the overview about software development practices methodologies, that will be via discussed the importance of used the methodologies among software developers. In the section 2.3 shows the agile software development practices methodology and an overview of agile methods practices. Section 2.4 focused on Extreme Programming. While, the adoption of agile methods issues were discussed in section 2.5.

2.2 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 (Kalermo & Rissanen, 2002; Chandra, Kumar, & Kumar, 2010). 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 (Blokdijk, 2014). 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 (Pressman, 2005; Chung, Nixon,

Yu, & Mylopoulos, 2012). 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 (Pressman, & David Brian, 2009; Pickering, 2001).

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 (McConnell, 2004; Miller, & Page, 2009; Cyganek & Siebert, 2011) 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 (Boehm, 2006; Unterkalmsteiner et al., 2012; Highsmith, 2013; Santos, 2014). 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 (Valacich, George, & Hoffer, 2009; Bird, 2007; Douglas, 2006).

According to Valacich, George, and Hoffer (2009) 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 (Valacich *et al.*, 2009). 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 (Cyganek & Siebert, 2011; Douglas, 2006).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 (Valacich *et al.*, 2009).

According to Leffingwell (2010) 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 (Stober & Hansmann, 2010; Cagle, 2010; Bustard, Wilkie, & Greer, 2013). Prototyping, time constraints, smaller project team members, management involvement, and iterative software development are all significant components of the agile software development process (Leau, Loo, Tham, & Tan, 2012; Eckstein, 2013). 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 (Stober & Hansmann, 2010; Cano *et al*., 2015).

2.3 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 (Shore & Warden, 2008; Leffingwell, 2010; Turk, France, & Rumpe, 2014). 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" (Shore & Warden, 2008; Stober & Hansmann, 2010; Soundararajan, Arthur, & Balci, 2012) and each of them consists of individual practices and techniques.

Many of the agile methodologies (then called as lightweight) were created in the 1990s (Sliger & Broderick, 2008; Stober & Hansmann, 2010) 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 (Larman, 2004; Petersen & Wohlin, 2009; Eckstein, 2013). NASA has used IID in software projects since the 1960s and IBM from the 1970s (Larman, 2004; Kruchten, 2013) and it has been promoted by several

software development thought leaders since the 1970s (Larman, 2004; Petersen & Wohlin, 2009). Also the ideas of Lean Product Development (used and propagated by Toyota in automobile production) have influenced the development of agile methodologies (Sliger & Broderick, 2008; Soundararajan *et al*., 2012) as they spread to North America and to the IT community at large in the 1980s (Aguanno, 2004). The actual term agile software development was coined in 2001 when 17 lightweight methodologists got together (Sliger & Broderick, 2008; Leffingwell, 2010) 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 (2001) the manifesto is also accompanied by the following 12 principles that reflect its four values as illustrated in Table 2.1.

Table 2.1 *Principles of the Manifesto for Agile Software Development(Fowler & Highsmith; 2001)*

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 (Stober & Hansmann, 2010; Leffingwell, 2010).

Moreover, the agile methodologies besides working and delivering in short iterations, an agile team works as a one sharing a common goal (Cohn, 2005). Problems are solved together regardless of roles. Documents are no longer passed from one specialist to another as the primary means of communication. Programming is seen as a comprehensive craft. Besides writing the code, it also includes the technical design (modeling) and testing of the code. Agile teams focus on business priorities by delivering complete user-valued features in customer

specified order to optimize the ROI (Cohn, 2005). Teams also include an on-site customer representative that works with the team daily to give feedback and define requirements for the software (Shore & Warden, 2008; Petersen & Wohlin, 2009). This interactive face-to-face communication (Cockburn, 2004) (and other practices) allows the team to develop the software without needing a detailed written documentation (like a traditional software requirements specification). If these stakeholders are not available, much of the agility of the project is lost as requirements need to be collected and documented by traditional means.

Furthermore, all agile software development methodologies include a number of different practices and techniques that define how requirements, design, development, testing and project management should be done. Many of the practices are shared between the different methodologies. In fact, there are more practices which could be introduced and summarize some of the more significant ones in Table 2.2.

Table 2.2 *Agile Practices and Methods (Cohn, 2005; Shore & Warden, 2008; Elssamadisy, 2008)*

Practice	Description	Work products
On-site customer	The whole team works together in a common project room with an onsite customer or a customer representative like a product manager. The on-site customer is a subject matter expert working on the requirements with the developers and is empowered to make decisions about the requirements. Having an on-site customer enhances communication helping to develop better quality requirements and reducing documentation overhead.	Story cards
User stories	User stories (or just stories) are a requirements engineering tool in agile projects. They are short written descriptions of features used for planning and as reminder, conversations to flesh out the details of the	Story cards

	feature and tests that document and determine that the feature is complete. They also include work estimation. User stories can be documented on paper index cards or in an appropriate software tool. In many ways user stories replace the traditional requirements documents.	
Project management		
Planning	Planning is done on several levels in an agile project and it usually means the process of creating, choosing and elaborating the next work items, such as user stories for example, for the next cycle (project, release, sprint, day). Planning usually involves a backlog which is a documented list of prioritized work items and their work estimates. Planning poker is a specific sprint planning technique in which the whole team participates.	Backlogs: • Product • Release • Sprint Burn down chart Story Cards
Small releases	The development progresses in a series of short, time boxed sprints, each producing new fully functional features and working software. Work cannot be added to a sprint once it has started.	Not applicable
Self organizing teams	During the sprint, the team itself is responsible for meeting the sprint's goals and has the authority to plan and execute its work as it sees fit. The role of the project manager is not to direct the team but to help it achieve its goals.	Not applicable
Sustainable pace	Work should be done at such a pace that can be sustained indefinitely. People create better, higher quality products when they are healthy, motivated and enjoying their work. Frequent overtime is discouraged as it causes opposite effects and is seen a sign of deeper problems.	Not applicable
Daily meeting	Each day a short meeting is held where the team coordinates its work, synchronizes daily efforts and assesses and revises its plans. The Daily Scrum (or stand-up) is specific technique related to Scrum.	Not applicable
Retrospective	Retrospectives are meetings held after sprints, releases and projects where the project team reflects its experiences and decides on possible actions for improving the process.	Not applicable
Design		
Metaphor	Metaphor supports the idea of having simple design by providing a frame of reference for how the team should think about the system and thus aiding communication about the design.	Not applicable
	Simple design means that the team aims to have the simplest possible design that is enough to deliver the	Not applicable

Simple design	features the customer needs. Simple doesn't necessarily mean simplistic but rather that the system should not contain anything unnecessary for its intended goal.	
Programming		
Refactoring	The program code should be continuously cleaned and restructured to achieve a better and simpler design without changing the external behavior / functionality of the system.	Program code
Pair programming	Pair programming is a practice where two developers program together in order to produce better quality code and to share an understanding of the code. The developers switch roles often, e.g., a few times in an hour and also pairs are switched on a daily basis.	Program code
Team code ownership	All code is owned by everyone and any pair of programmers can change any code. This collective responsibility encourages problems to be fixed as they are spotted and development is faster as the "bottleneck" of individual code ownership is removed.	Not applicable
Sustainable pace	Work should be done at such a pace that can be sustained indefinitely. People create better, higher quality products when they are healthy, motivated and enjoying their work. Frequent overtime is discouraged as it causes opposite effects and is seen a sign of deeper problems.	Not applicable
Testing		
Continuous integration	All code checked in to a version control system is automatically and continuously re-integrated and tested on a separate build machine. Doing integration testing continuously and automatically lets the team find related problems when they are created and reduces the amount of repetitive manual work needed to integrate the software.	Automation Scripts
Test-driven development	In test-driven development a unit test is written before the actual program code that passes the test. Testing can be automated by writing test code that runs the actual operational code. This approach helps to create a comprehensive set of tests for the system and ensures good code quality.	Test code
Sprint review	At the end of each sprint, a meeting is held where the results of the sprint are demonstrated to stakeholders in the form of working software. The goal is to share information, evaluate the design and functionality of the software, get feedback and brainstorm future directions.	Not applicable

Customer acceptance testing	Customer representatives write acceptance criteria for the user stories they create which then can be automated by the developers. These tests are then run to see if the related feature was developed as defined in the user stories.	Story cards Test code

In the same context, there are several agile methodologies used by the developers, but the most notable methodology is Extreme Programming (XP) (Alite & Spasibenko, 2008). Figure 2.1 shows the flexibility (how they accept change) and quality (defects and accuracy of the product) of XP method compared to other software development.

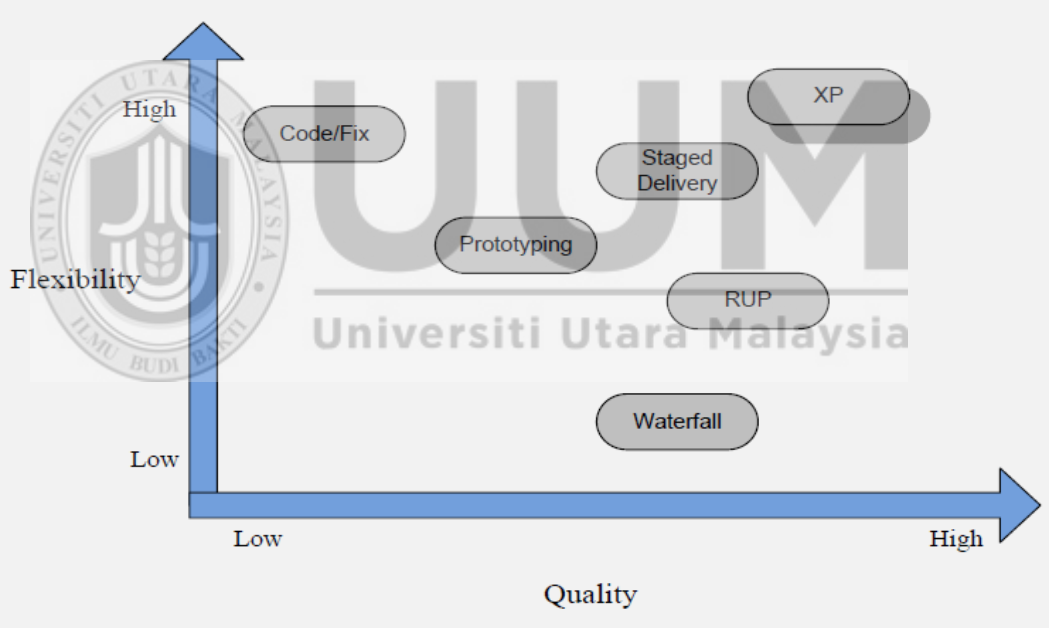

Figure 2.1 Comparison of the Methodologies(Baird; 2002)

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.

2.4 Extreme Programming Practices (XP)

XP was created by Kent Beck and Martin Fowler (Wells, 2009) while working for Chrysler Corporation and was first published in his book (Beck, 1999) .The name reflects the idea that teams should take good, proven engineering practices to the extreme (Sliger & Broderick, 2008). XP stresses "*customer satisfaction through rapid creation of high-value software, skilful and sustainable software development techniques and flexible response to change*" (Larman, 2004). According to Shore and Warden (2008) 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 (Rittenbruch, McEwan, Mansfield & Bartenstein, 2002; Darwish, 2013;):

1- **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.

2- **Feedback**: developers must always have a way for obtaining information regarding the development process. Feedback includes several dimensions:

the system, customer, and team members. Feedback from the system and the team members aims to provide project leaders with quick indicators of the project's progress whereas feedback from customer includes the functional and acceptance tests.

3- **Simplicity**: is one of the values supported explicitly by XP. A simple design always needs less time to finish than a complex one. Therefore, XP encourages developers to start with the simplest solution. Extra functionality can then be added later. Programmers do the simplest thing that could work, and leave the system in the simplest condition. This improves the general speed of development while still retaining an emphasis on working software.

4- **Courage**: XP encourages the team members to make decisions that support the implementation of XP practices. The team members need courage to refactor the software code. The team members review the existing system and modify it to facilitate the implementation of future changes. In addition, courage may include removing parts of source code that is obsolete, no matter how much effort was used to create these parts.

The emphases of XP's practices are on programming and the quality of code, but it is also a communication and team oriented methodology. XP does not require other detailed work products (like a requirements specification document) but program code and test cases. Oral communication is the suggested way of working with requirements and design. The whole team, including customers, developers and managers, is expected to work together in the same project space to quickly deliver software with high business value (Martin, 2003; Stober & Hansmann, 2010).

The customer's role in an XP project is to document software requirements/features as user stories, prioritize these stories by their business value and write and execute tests that demonstrate that the stories are implemented as expected. The XP programmer role is versatile making no distinction between programmers, designers, testers and so on. All programmers work as a team and share responsibilities that might be assigned to specific individuals in a non-XP project. In addition to the design and development tasks, the programmers are responsible of making work estimations for the user stories and writing automated unit tests for everything they program. The team might also have an XP coach or a project manager who monitors the use of XP practices and keeps the work ongoing (Leffingwell, 2010). In addition to the values and principles, XP includes twelve software engineering practices which it combines for greater synergy as shows in Figure 2.2.

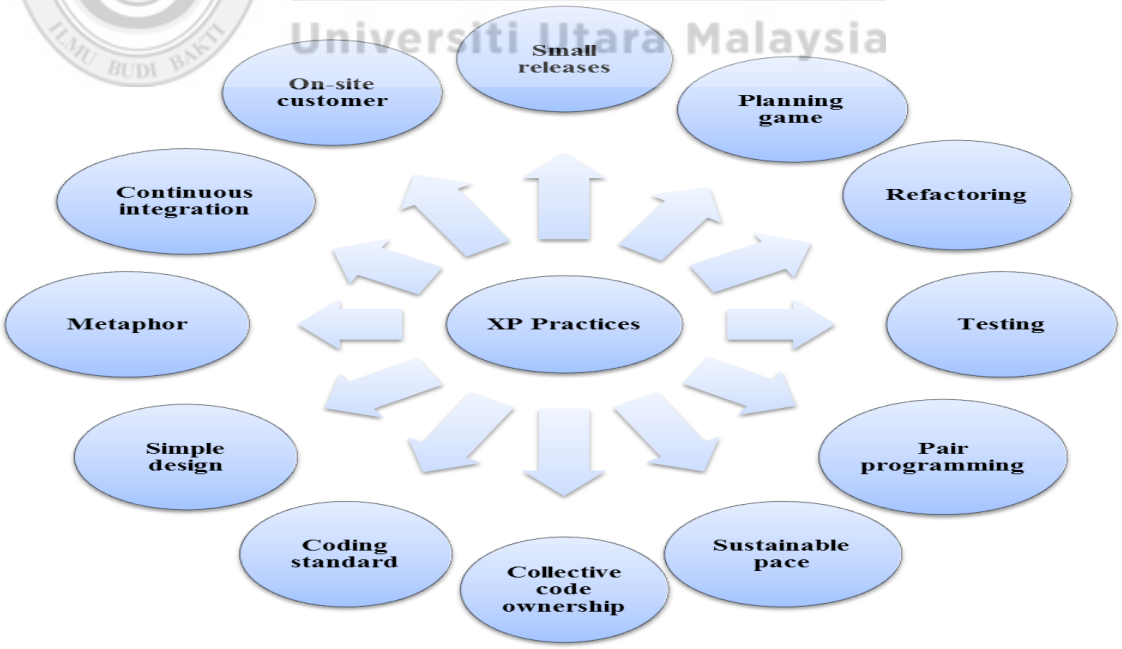

*Figure 2.2*Original XP practices (Cohn, 2005; Darwish, 2013; Turk, Franc & Rumpe, 2014)

i. **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 (Syed-Abdullah, Omar, Hamid, bt Ismail, & Jusoff, 2009; Wood, Michaelides & Thomson, 2013).

ii. **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 (Jun, Qiuzhen & Lin, 2010; Abrantes & Travassos, 2011).

iii. **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 (Lindstrom & Jeffries, 2004; Turk *et al*., 2014b).

iv. **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 (Aveling, 2004; Mushtaq & Qureshi, 2012; Omar, Abdullah & Lailee, 2013).

v. **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(Salo & Abrahamsson, 2008; Wood *et al.*, 2013).

vi. **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 (Sison & Yang, 2007; Abrantes & Travassos, 2011).

vii. **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 (Kongyai & Edi, 2011; Hummel, 2014).

viii. **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 (Begel & Nagappan, 2008; Rumpe & Schröder, 2014).

ix. **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 (Lemos, Ferrari, Silveira & Garcia, 2012; Turk *et al*., 2014).

x. **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 (Harriosn, 2003; Singhal & Banati; 2014).

xi. **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 (Siebra, Mozart Filho, Silva & Santos; 2008).

xii. **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 (Maurer & Martel, 2002). 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 (Jeffries, 2003).

Based on the discussion above and the previous studies, Table 2.3 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 2.3 *XP practices mapping with respect to quality subjects (Dubinsky & Hazzan, 2002)*

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

2.5 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 Leffingwell, (2010) 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 (Aguanno, 2004). 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." (Poppendieck & Oppendieck, 2003). 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 (2005) 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.

2.5.1 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 (Marrington, Hogan & Thomas, 2005). However, some researchers have reported that large-scale and complex projects have benefited from suitably tailored agile development methods (Bowers, May, Melander, Baarman, & Ayoob, 2002; Lippert *et al*., 2003; Cao, Mohan, Xu, & Ramesh, 2004; Lindvall *et al*., 2004).

As well as, Bowers *et al* (2002) 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* (2003) 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 (Beck, 2000).

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* (2003) 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 (2006), 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 (Beck, 2000). Scrum focuses on project management of iterative development (Schwaber & Beedle, 2002), and Adaptive System Development (ASD) is a framework for managing software projects under intense time pressure (Highsmith, 2000).

2.5.2 Strengths and Weaknesses of XP Method

Many researchers indicate the strengths and weaknesses of XP method. Table 2.4 depicted these cons and pros of the XP based on number of the researchers.

Table 2.4 *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.
XP method saves the project against the cancellation with the help of periodic releases.	Beck, 2000; Guha *et al*., 2011.
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.
XP method is more flexible and includes more explicitly the needs and intentions of all project participants.	Beck, 2000; Fruhling & Vreede, 2006; Xu, 2009.
By test driven development practices, XP method resulting in less errors and acceptance of changing requirements.	Beck, 2000; Fruhling & Vreede, 2006; Munassar & Govardhan, 2010.
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.
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.
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.
Lack of project management practices.	Beck, 2000; Turk *et al*., 2002; Mushtaq, 2012.
Lack of documentation though the development lifecycle.	Qureshi, 2011; Munassar & Govardhan, 2010; Guha *et al*., 2011; Paulk, 2001.

Developers must be experienced.	Paulk, 2001; Munassar & Govardhan, 2010.

2.6 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 (2012) 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 2.5.

Table 2. 5 *Summary of the Application Extreme programming Practices*

Authors	Year	Type of the study	Finding
Sfetsos, Angelis & Stamelos	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	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.	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	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	2013	Qualitative	• The results indicated that software development using XP method delivered quickly.
Mohamed, Farvin, Baharom, & Deraman,	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	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.

2.7 Summary of Chapter Two

The literature review is important for clarifying the problem statement and also to understand the elements related to phenomena. Therefore, this chapter focused on the agile method as a preliminary introduction to the XP method. Following that, XP method with all the pertaining components discussed. Finally, several of previous studies have been included in the current study as a related work.

Twelve XP practices discussed in more details to understand the content each practice to achieve the first objective of this study. As well as, literature review help the research also to achieve the last research objective through extract the codes and themes from the interviews. In this chapter also highlighted several strengthens and weaknesses of the XP approach. Also the prior literature asserted to need more empirical studies related to XP practices in Asia and especially in Malaysia. In the next chapter, the process to attain the aim of the current study will be discussed in details.

CHAPTER THREE
RESEARCH METHODOLOGY

3.1 Introduction

This chapter includes a detailed description of the research methodology that was utilized in the study. Methodology is a research process that applies a collection of methods to resolve the problem (Ishak & Alias, 2005; Pickard, 2012; Flick, 2015), which is organized in phases, and each of phase has been completed sequentially. However, the aims of the research process is to achieve the objectives of the study. This chapter will be a description of the research design and method used to address the research problem as outlined in chapter one. The remainder of this chapter will discuss the data collection procedures that include the details on the technique to be used, which include: literature review and interview. Following this data analysis and interpretation of findings will be discussed.

3.2 Research Design

The research design differs depending on the methodology. According to Creswell (2013), there is no definite structure to design a qualitative study. Generally, the research design encompasses tasks such as sample design, data collection design, and methodology tests. The type of problem can influence the choice of the methodology. According to Creswell (2009), research designs are plans and the procedures for research that span the decisions from broad assumptions to detailed methods of data collection and analysis. Hence, it implies the plan for conducting the study (Creswell, 2013).

Basically, there are two methods of data analysis: qualitative and quantitative analysis (Sekaran & Bougie, 2010; Zikmund, 2003), where most researchers prefer to utilize either quantitative or qualitative (Ragin, 1987; Kaplan & Duchon, 1988; Lee, 1991; Gable, 1994; Mingers, 2001). Previous studies also show that qualitative methods are very famous for IS studies. Puvenesvary, Rahim, Naidu, Badzis, Nayan, and Aziz (2008, p.1) say that: "*An extensive literature review in your area of study will help you determine the most appropriate method to use in your research*". Similarly, Saunders and Lewis (2012), state that the previous studies will give the researchers ideas about how they might collect data. In addition, if a concept or phenomenon needs to be understood because little research has been done on it, then it merits a qualitative approach (Creswell, 2009). Hence, in this study, in order to investigate Extreme Programming Practices at UUM IT, the qualitative approach is used.

This qualitative study seeks to develop a deep understanding on the practices used in UUM IT. The researcher collected extensive data from experts who work in UUM IT because individuals with more experience in the central phenomenon enable the researcher to obtain more in-depth data (Creswell & Clark, 2007). Furthermore, research design expresses a plan for the study, providing the overall framework for collecting data, outlining the detailed steps of the investigation, and providing guidelines for systematic data collection (Lankshear & Knobel, 2004; Creswell, 2006). The research design can also be described as the specific procedures involved in the last three steps of the research process: data collection, data analysis, and report writing (Alison, 2000; Creswell, 2005). In regards to the research design as

defined by those authors, the research design for this study is summarized in Table 3.1

Table 3.1 *Overview of research design and methodological processes*

ASSUMED PARADIGMS	
Methodological Approaches	Qualitative Case study Approach
SELECTION OF PARTICIPANTS	
Purposive sampling	With the assistance of the Director of UUM IT, five of the UUM IT experts who have experience in this phenomenon have been selected.
DATA COLLECTION	
Data collection instruments	Individual interviews
DATA ANALYSIS AND INTERPRETATION	
Transcription, data coding, free quotations, and links	

3.3 Research Approaches

The qualitative approach is utilized in this study. Creswell (2009) recommends that qualitative approach is able to explore the complex set of factors surrounding the central phenomenon and present the varied perspectives or meanings that participants hold. Furthermore, Figure 3.1 illustrates the overview of this study which involves research process activities to achieve the objectives of this study.

The initial phase of the research process involves problem identification and building the conceptual model for the best practices depending on the Extreme Programming (XP) Method. Extreme Programming (XP) is a software engineering methodology and the most prominent of several agile methodologies (Abrahamsson, Warsta, Siponen & Ronkainen, 2003). In addition, Siebra, Mozart Filho, Silva and Santos

(2008) state that XP encourages particular values such as short quick development steps, feedback, communication and adaptation to clarify the requirements and design. However, during this phase, an extensive literature survey is conducted to understand the problem and XP practices. Although this phase only involved preliminary and basic work, the foundation of the study is very essential in order to ensure that the researcher has defined the problem to be studied correctly and to make sure the researcher is heading in the right direction for building the conceptual model. Thereafter, based on the prior literature the researcher will attain the first objective *"To identify the best practices of Extreme programming (XP)"* from this objective the researcher will highlight the XP practices and draw the suitable conceptual framework. Achieved these objectives first is very important to the present study. Where, by these attained these objectives the researcher can conduct the interviews with the participants (experts) and also understanding the effect if ignore one of these practices on the quality of the software.

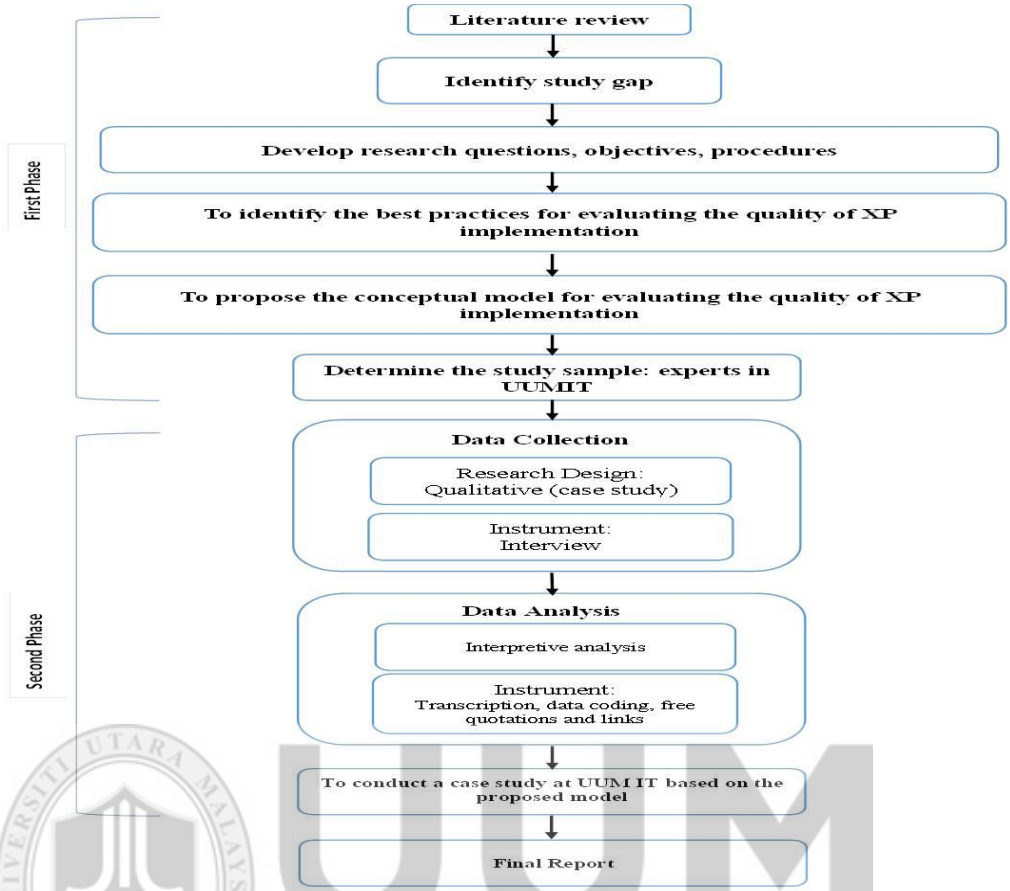

Figure 3.1 Research Process of the Study

The second phase of the research process involves gathering data from the participants once the instrument has been developed and the sample of the current study has been identified. The data was collected primarily through semi-structure interviews with five experts in UUM IT to attain the last objective *"To validate the proposed conceptual model using a case study at UUM IT"*. After the required data was collected, the data analysis procedures were then performed. Then, interpretive analysis was applied as the data analysis technique in this study. In this phase mainly strive to explore any of the XP practices are applied among the UUM IT team and why other practices not apply or partially apply. The next sections of this study further discuss the research elements mentioned above.

3.4 Data Collection

Based on the general characteristics of qualitative research, qualitative data collection consists of collecting data using forms with general, emerging questions to allow the participants to generate responses; gathering word (text) or image (picture) data; and collecting information from a small number of individuals or sites (Creswell, 2012). In addition, in qualitative inquiry, the intent is not to generalize the results to a population, but to develop an in-depth exploration of a central phenomenon (Creswell, 2012; Baker & Edwards, 2012; Maxwell, 2013).

3.4.1 Sampling

With regard to sampling, Roulston (2010, p.81) says that *"researchers include participants in studies on the basis of ease of access or ready availability are using convenience sampling"*. Furthermore, in qualitative research, the researchers intentionally select participants who have experience with the central phenomenon or key concept being explored (Creswell & Clark, 2007; Creswell, 2012, 2013).This study sampled (experts) the subjects among those who have experienced working with IT development in UUM IT. Moreover, most of these experts have experience with UUM IT (formerly called computer center) for more than 10 years. Regarding this study, there are 5 experts who have accepted to participate in the semi-structured interview.

3.4.2 Research Instrument

Many common qualitative research instruments can be used to collect qualitative data, including participant observation, interviews, and focus group interviews

(Tsvara, 2013). Among them, this study adopted the distinct instrument, particularly the semi-structure interviews. This type is particularly suited for obtaining specific data. Semi-structure interviews are operational for collecting data on individuals' personal perspectives, perceptions, and experiences, particularly when sensitive topics are explored.

In this study, interviews were used to gather data from the employees as suggested by a number of previous studies (Marshall & Rossman, 1999; Fontana & Frey, 2005; Chism, Douglas & Wayne, 2008). Technically Ackroyed and Hughes (1992, p.100) state that "*interview encounters between a researcher and a respondent in which an individual is asked a series of questions relevant to the subject of the research*." In other words, a qualitative interview occurs when researchers ask one or more participants general, open-ended questions and record their answers (Creswell, 2012). Willig and Stainton-Rogers (2007) argue that interview is one of the most powerful and widely used tools for qualitative research. The questions for this study are organized in several sections, where each section is related to the specific XP practice. Table 3.2 below listed the questions that exploited to extract the raw data from the experts through the interview session.

Table 3.2 *Interview Questionnaire*

XP PRACTICES	QUESTIONS	References
On-site customer	How many times you can get feedback from your client? How do you get feedback from your client? How long it will take when you see your client? Does your client always give immediate and consistent feedback? Does your client helpful? e.g. give sample code, clear requirements Does your client always change his/her requirements? How many requirements changes? How did you get clear requirements?	Williams, Layman and Krebs (2004); Sfetsos, Angelis and Stamelos (2006); Begel and Nagappan (2007); Manyam and Kurapati (2011); Qureshi (2012); Jalali, Wohlin and Angelis (2014); Harris (2014)
Planning game	Do you write user requirements? Do each member responsible for each story cards? Do you discuss the requirements with your client? Does your client write user requirements? Do you estimates the time needed to complete the task in user requirements? Do you prioritize the user requirements? How did you prioritize the	

	user requirements? How you documents the requirement gathered?	
Collective code ownership	Can any of your team change code that he/she did not originally write? how often do they do so? Does your team have a code repository? How did you manage the repository?	Williams, Layman and Krebs (2004); Sfetsos, Angelis and Stamelos (2006); Begel and Nagappan (2007); Manyam and Kurapati (2011); Qureshi (2012); Jalali, Wohlin and Angelis (2014); Harris (2014)
Coding standard	Do you have and adhere to team coding standards? How often it is followed? What type of coding standard you are followed? There was hardly any degradation of code due to difference in coding standards between two XP partners working on the project? If No, how the standard maintain? Does the programmers worked efficiently in the presence of proper coding standards? Why?	
Continuous integration	How often you synchronize and check in your code? Which tool used to assist source code integration?	
Frequent release	When is the first release?	

	The early release give the team an opportunity to improve which of the following areas? All the reviews (release) is being tested thoroughly? Do early reviews (release) help in fixing bugs better? How? Why?	
Sustainable pace	Did the first release cover most of the functionalities mentioned in the specification document? What percentage of the estimated functionalities was captured? How many releases were made before the final releases? How well do you pace yourself? Do you delivered a completed project?	Williams, Layman and Krebs (2004); Sfetsos, Angelis and Stamelos (2006); Begel and Nagappan (2007); Manyam and Kurapati (2011); Qureshi (2012); Jalali, Wohlin and Angelis (2014); Harris (2014)
Pair programming	How easy do the team members accept pair programming? What are the criteria that decide the partners? How often your works done in pair? How often do you swap between partners? What criteria when you decide to swap partners? State any difficulties during pair programming implementation?	
Test first programming	When were the test cases developed?	

	Is it easy to develop test cases before coding?	
	What percentage of the time you employ test-first programming?	
	Does the client help with the test cases?	
	All codes undergo unit testing?	Williams, Layman and Krebs (2004); Sfetsos, Angelis and Stamelos (2006); Begel and Nagappan (2007); Manyam and Kurapati (2011); Qureshi (2012); Jalali, Wohlin and Angelis (2014); Harris (2014)
	Did you use any automated tool to unit testing?	
	How often functional testing is carried out?	
Simple design	Did you do the simplest thing that can possibly work?	
	How often do you succeeded in 'keeping it simple'?	
Refactoring	Does the team reuse code? How often?	
	Does reuse code can help to speed up the development process?	
	How? Why?	
	How often do you stop to cleanup code that has already been implemented without changing functionality?	
Metaphor	How often do you feel that this is true of the system develop? For instance, classes and methods have good descriptive name, other members do not need often to ask and refer to understand architecture, and the client understand and explain the metaphor.	

3.5 Data Analysis and Interpretation

Qualitative research is "interpretive" (Dey, 1993; Creswell, 2012) research, in which a personal assessment to a description that fits the situation or themes that capture the major categories of information is made. This implies that data analysis in qualitative research consists of preparing and organizing the data for analysis, then reducing the data into themes through a process of coding and condensing the codes, and finally representing the data in figures, tables, or a discussion (Creswell, 2007; 2013). In addition, qualitative data analysis is a range of processes and procedures whereby we move from the qualitative data that have been collected into some forms of explanation, understanding, or interpretation of the people and situations we are investigating (Creswell, 2005; Lewins, Taylor & Gibbs, 2005).

However, there is no single, accepted approach to analyzing qualitative data, although several guidelines exist for this process (Dey, 1993; Miles & Huberman, 1994). It is an eclectic process because each qualitative study is unique, so the analytical approach used will be unique (Saldaña, 2012). With reference to the discussions in the previous paragraphs, this study adopts the several steps in analyzing and interpreting qualitative data by Creswell (2009; 2012).

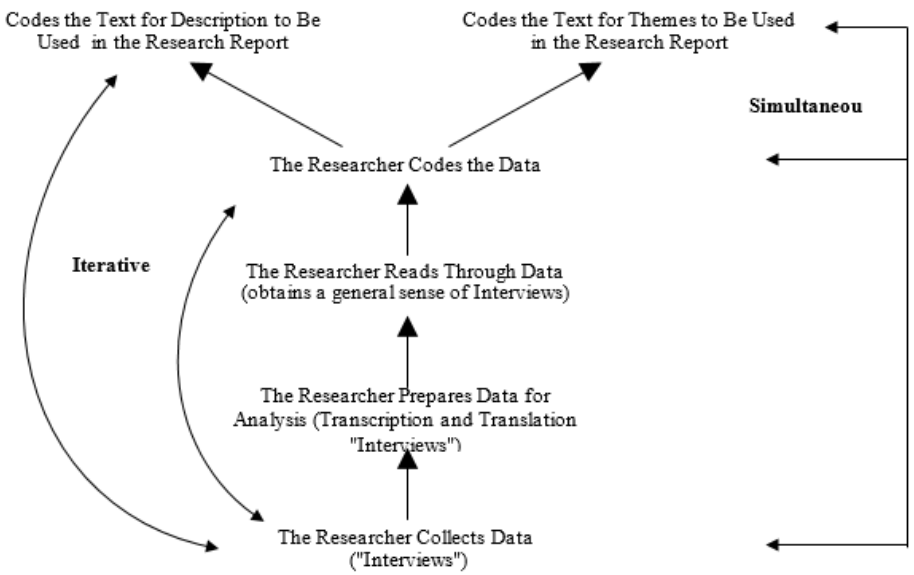

Figure 3.2 The qualitative process of data analysis

As seen in Figure 3.2, this study first collected data and then prepared them for data analysis. This analysis initially consists of developing a general sense of the data and then coding description and themes about the central phenomenon. Moreover, the twelve XP practices will be used as a guide for coding the interview data.

Furthermore, with the popularity of computers, researchers have a choice whether to manually analyze the collected data or to use a computer (Creswell, 2012). This study uses Nvivo 11 (Figure 3.3) to organize data or themes. This is also influenced by the fact that Nvivo can facilitate the qualitative research process by making all investigation phases open to public inspection (Sinkovics & Alfoldi, 2012).

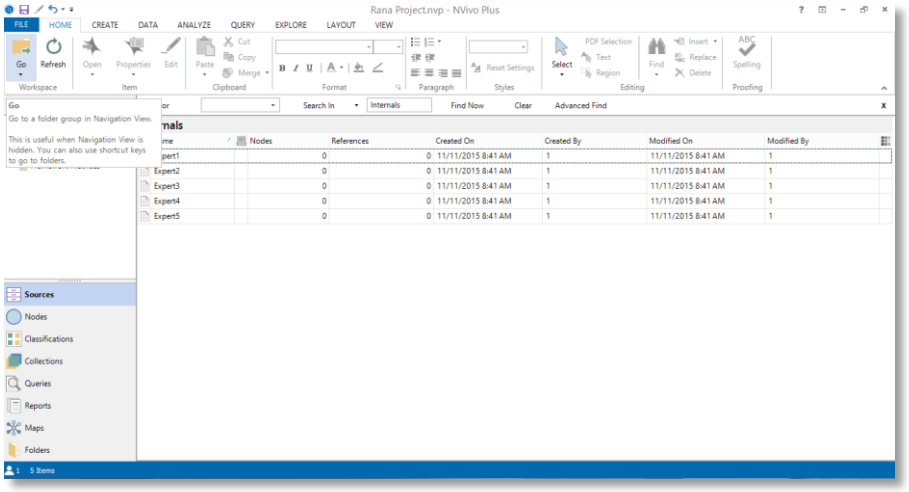

Figure 3.3 Nvivo project

3.6 Validation of Data Collection

Member checking technique strives to harness validation in this study. This technique has been considered by Lincoln and Guba (1985, p.314) as "*the most critical technique for establishing credibility*." Member checking is a process in which one asks one or more participants to check the accuracy of the account (Creswell, 2012; Yin, 2011). This does not mean taking back the raw transcripts to check for accuracy; instead, parts of the polished product, such as the themes, the case analysis, the grounded theory, the cultural description, and so forth (Creswell, 2009) are re-analyzed. Once an analysis is completed, the analysis (the findings and specific description) will be returned to the participants for confirmation of accuracy (Yin, 2011). In this research, two experts accepted to check the findings, and they agreed with the final results (see Appendix B).

3.7 Summary of Chapter Three

This chapter discusses the research design and methodology in preparation for the empirical investigation utilizing a qualitative method in the study of discovering the XP practices in UUM IT. At the same time, it evaluates the proposed conceptual model. More specifically, the research design and research approach are discussed in detail. It also describes the methods of data collection, interviews, and data analysis processes. The findings from the study will help determine if there is lack of use of XP practices in UUM IT. Also, the current study seeks to shed light on the practices that need more focus to enhance the performance of UUM IT.

CHAPTER FOUR
DISCUSSION OF RESULTS AND FINDINGS

4.1 Introduction

Chapter 3 states that the primary method of data gathering is individual interviews, the majority of which were semi-structured. In total, there are 5 semi-structured interviews conducted with experts to gain the employee's viewpoints regarding the XP practices that harness in the UUM IT. Whilst, the present chapter a subdivided into three several sections. In the section 4.2, the XP practices were discussed to achieve the first objective: *"To identify the best practices for evaluating the quality of Extreme Programming (XP) implementation.* "Then based on the first objective, the researcher visualized the second objective: *"To propose the conceptual model for evaluating the quality of Extreme Programming (XP) implementation."* To achieve the third objective:*"To conduct a case study at UUM IT based on the proposed model."* As aforementioned, the interview with experts conducted for this purpose. In the next sections, these objectives debated and presented in more detail.

4.2 XP Best Practices

Nowadays, agile software development has become a common way of developing software, especially in the information systems domain. According to Pressman (2009), Abrahamsson, Salo, Ronkainen and Warsta (2002), and Xu, Lin and Foster (2003**),** one of the popular agile methods is Extreme Programming (XP). Where, Extreme programming (XP) embraces both communication and feedback as interdependent process values which are essential for projects to achieve successful

results. Extreme Programming (XP) is a lightweight software development method that has got its popularity because of its best practices (Nawrocki, Jasiński, Walter & Wojciechowski, 2002).

In addition, XP has introduced new way of software development and is efficient, low risk, welcome changes, predictable, scientific and is different from other methods because of strong oral communication, pair programming, automated test, collective code ownership and introduced story telling culture (Beck, 1999). As well as, Hussain, Lechner, Milchrahm, Shahzad, Slany and Umgeher (2008) show that most of the practices in the XP methodology can be used directly in the project while some required little changes according to the environment. According to Beck (2000) and Jeffries, Anderson and Hendrickson (2001), it can be concluded that the XP consists of twelve practices as follows:

Table 4.1 *XP Best Practices*

Xp Best Practices
On-site customer
Planning game
Collective code ownership
Coding standard
Frequent releases
Continuous integration
Pair programming
Test first programming
Sustainable Pace
Simple design
Refactoring
Metaphor

i. **On-site customer**: a customer needs to be available to determine and prioritize the requirements. This is one of the few requirements in XP and it

helps to improve the software business value. However, the programmers can get input from the customer immediately instead of speculating. Quick changes to the focus of the development can also be made when necessary.

ii. **The Planning Game:** this practice means a set of rules and moves that may be used to simplify the release planning process, and it is closed interactions between customers and programmers.

iii. **Collective Code Ownership**: everybody in a XP project takes responsibility for the code in the whole system. Any improvements or new ideas can be added anywhere in the code, where this can be made partly due to the automated tests in XP. Moreover, unknown repercussions will be detected by the automated tests and the programmers can modify the code more freely. Therefore, this practice increases quality of the code and reduces faults.

iv. **Coding Standards**: coding rules exist and are followed by the programmers. Therefore, this practice keeps the code consistent and easy for the entire team to read. Re-factoring and all the codes in the system look coherent and harmonious. Furthermore, this practice helps the XP team to understand all the codes that have been written as basis for the practice of collective ownership.

v. **Frequent Releases:** this practice means all releases should be a small as possible, but with the maximum quantity of business features developed, whereas short cycles are used to reduce the risk when a project fails to produce business value to the customer, and also helps in reducing planning problems and the problem with changing requirements during the development process. Moreover, frequency is important as well depending on

which kind of software is delivered. At the end of every iteration; software is visible, and given to the customer.

vi. **Continuous Integration**: changes to the code are integrated at least once a day. The pair programmers are responsible for integrating their own code and automated tests are run to ensure that the system is working at 100 %. If the tests fail, the pair can undo their changes and start over. Therefore, this practice keeps the system never far from a production state. Moreover, the pair should check that their changes do not affect another part of the system developed by another pair of programmers. In addition, one machine can be used only for integration issues for one pair of programmers.

vii. **Pair Programming:** the production of codes is written with two people using one computer. One of them has control of the keyboard/mouse and creates the code, and the other is continuously assuring quality by watching, trying to understand, asking questions, looking for alternative approaches, and helping to avoid defects. If pairs are switched through the team knowledge is shared to everyone working in the XP team. Therefore, individual's skills are improved because the pair should switch at least once per day.

viii. **Test first programming**: testing is an essential part of XP; especially the automated tests, a feature without an automated test does not exist. In this practice the programmers write the unit tests and the customer writes the functional tests. This practice can be divided in two parts. First: Programmer Tests: programmers should create the tests first and then code. The first test should fail, because no codes have been created, and then the programmers

should create the code to pass the test, and then turn the cycle to add one more test followed by the code. One of the benefits of extreme programming is that 100% of the code is tested. While, in the second part, Customer Tests: each user story that represents a feature in the XP development has an associated acceptance test that is determined by the XP customer and implemented by the team. Moreover, the correctness of the systems is shown to the customer when all tests are passed. Consequently the application is continually growing and evolving.

ix. **Sustainable Pace (or 40 hours week):** this practice means that the team members work hard at a pace that they can go along with for the time being. However, overtime is a symptom of a serious problem in an XP project.

x. **Simple Design:** the design should be kept simple through the developments, using the developer's test-driven development and refactoring, whereas XP fits the design for the present system features ready for future changes in an incremental or iterative way. Therefore, XP design should begin without thinking of infrastructure, where the right design in XP can run all the tests, has no redundancies, and has the fewest possible classes and methods Moreover, XP focus on solving today's problems and every piece in the design must be able to justify its existence.

xi. **Refactoring**: refactoring is a process of changing a software system in such a way that it does not alter the system behavior of the code yet improves it internal structure. Doing design improvement in an XP project is a practice where the programmers delete duplicate codes. In addition, programmers should increase cohesion and decrease coupling. Therefore, refactoring

should be made when there is something wrong in the code, such as: classes that are too long, methods are too long and duplicate codes. Moreover, design improvement should be done every hour or half hour, followed by testing of what was done and this is done to keep the design as simple as possible at all times. Accordingly, the changes of the structure are verified with automated tests which help the programmers to get feedback on the changes.

xii. **Metaphor**: both the customer and the programmers share a story based on a metaphor that guides all development by describing the functionality of the system. Additionally, the team shares some common understanding from their past experiences. A metaphor should help everyone on the project to understand the basic elements and their relationships, where metaphor is similar to what other people call "*an architecture*", but with the addition that requires the XP team to follow some way of cohesion.

4.3 Proposed Conceptual Model

Miles and Huberman (1994) define a conceptual model as a visual or written product. Meanwhile, Mills (2010) believes that conceptual model uses deductive research to produce general information about relevant issues of a study (literature review). Thus, an inductive research is often carried out, and focusing on an in-depth analysis of these relevant research topics (interviews).

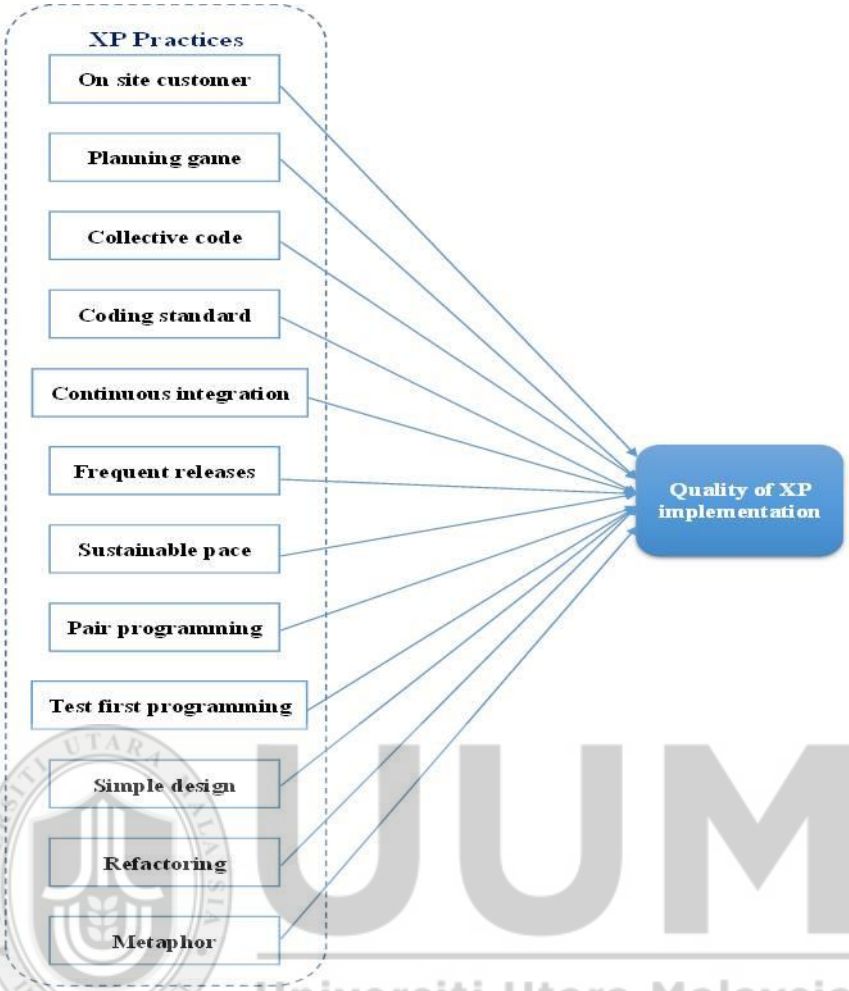

Figure 4.1 Conceptual Model of XP Quality implementation

Based on the best XP practices (see first objective) identified, a conceptual model was developed as in Figure 4.1. This figure visualized the main twelve practices that effect on the software quality when exploited XP approach. Most of the prior literature suggested to study all these practices when need focus on the quality of the XP approach. In fact, this conceptual model can help the researcher to highlight the fully adopted practices and also the partial and not adopted practices.In the next phase, the interviews were analyzed based on each practice (see appendix A).

4.4 Case Study Results at UUM IT with Five Experts

In this study, five experts were participated in order to get in-depth understanding of XP implementation in UUM IT, Table 4.5 listed the personal information for each participant. The XP quality implementation was evaluated based on their experts' opinion.

Table 4.2 *Expert's profile*

Experts	Position	Expert Experience	Location
Expert1	System analyst	15 years	UUM IT Center
Expert2	System analyst	26 years	UUM IT Center
Expert3	System analyst	20 years	UUM IT Center
Expert4	Programmer	18 years	UUM IT Center
Expeet5	System analyst	12 years	UUM IT Center

Figure 4.2 depicts the Nvivo's results. In the following sections, the researcher was analyzed the interview session according to the 12 practices and also the quality of the XP method.

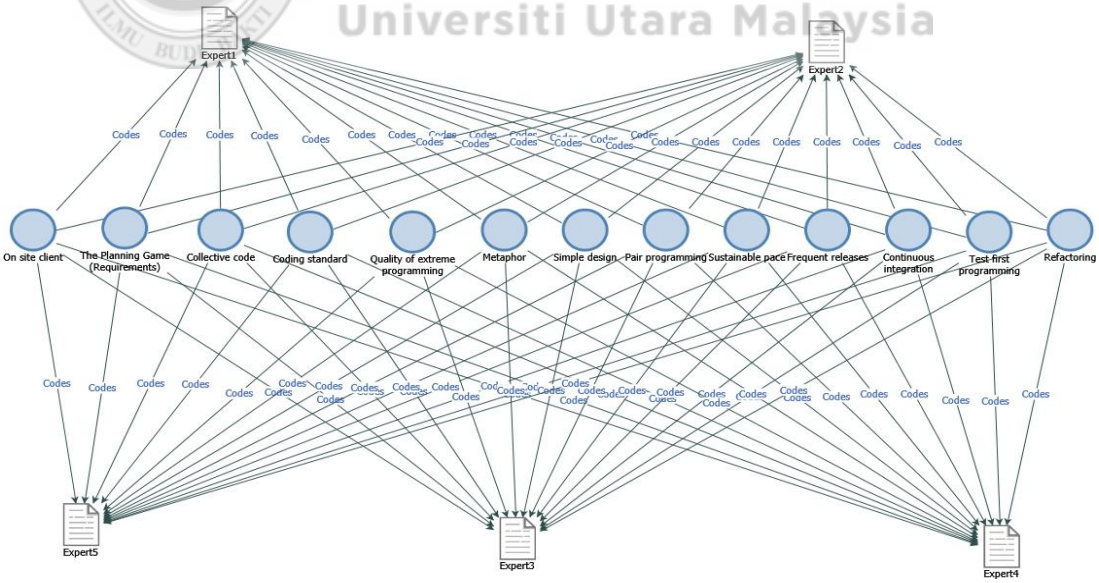

Figure 4.2 The interview based on the themes (Nvivo 11)

In Figure 4.2 above, illustrates the final outputs for all the participants based on the XP Practices as a theme. All the participants asked same semi-structure questions related to each practice. In the follow, will discuss each expert separately based on twelve practices of XP approach.

4.4.1 Expert 1

In fact, several questions have been asked to the participants related to the main purpose of the current study. These questions categorized based on the XP practices. The first group of the questions in fact was related to the *"On-site customer".* In this practice,the customers' job is to write and prioritize requirements, assist with acceptance testing and be on hand to answer questions from the development team as they arise. With regard to this issue, our participant (Expert 1) highlighted several elements pertaining to this practice, such as she illustrated the period for get the feedback from the customers (or also can called client): *"Frankly, we get the feedback from our clients in some times daily or weekly"* this period (daily or weekly) between the customers and the developer can increase the communication.

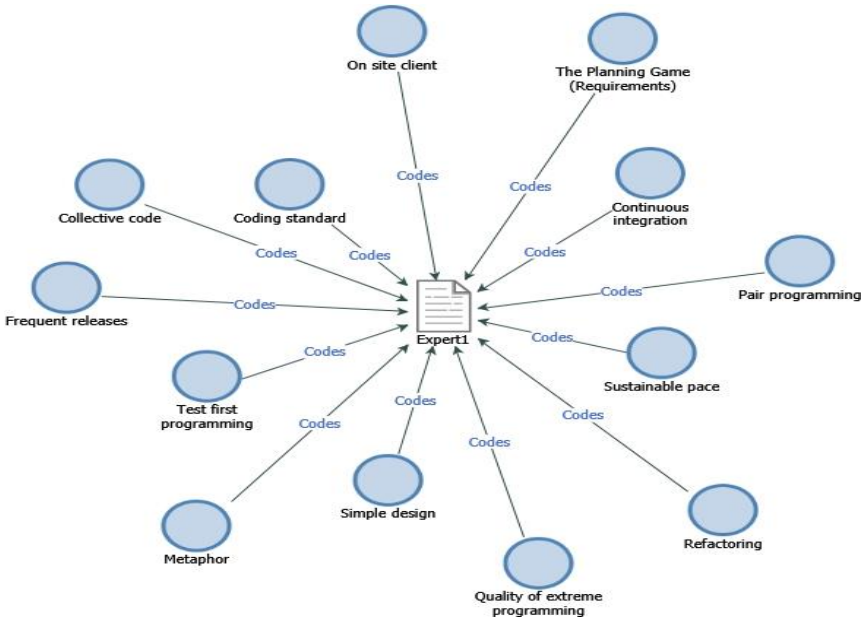

Figure 4.3 Expert 1 with XP practices

As well as, the participant (Expert 1) said that: *"the modern technologies such as, 'email', help us to get the feedback information from the customer* "Besides, and related to the respond by the customer, the programmers can get customer input immediately rather than speculate on customer preferences. And this confirmed by expert 1, who pointed: *"our client always give immediate and consistent feedback"*While, regarding to customers change requirements, our participant also referred that *"The client can always change his/her requirements and he/she always change less than three requirements."* Applied this practice can dramatically improve both developer productivity and the software's business value.

The second practice is the Planning Game (some authors also called Requirements). This practice closed interactions between customers and programmers. The expert was asked a number of the issues related to Planning Game to ensure UUM IT apply this practice or not. Such as, the expert asserted: *"we are always discussing the*

requirements with our clients to get clear requirements" on top of that, the expert also mentioned that, the client also is able to write the requirements. Where, she said that: "*The client can write the user requirements*" this practice also include negotiate about the scope and date between the customers and developers based on resources and business values. And this also stated by expert 1, who said that: *"In our work we always give priority the user requirements based on client need and urgency"* and also she referred that, *"I always estimates the time needed to complete the task in user requirements".*

The third XP practice under investigation was collective code ownership. The goal of the practice is to ensure that all developers collectively own the code to be able to make changes and that a loss of a small set of programmers does not lead to project failure. This character of this practice stated by our participant (Expert 1), who pointed that, *"Any member in my team can change code that he/she did not originally write but not all the time"* While, related to the code repository, the Expert 1 also highlighted that *"My team have a code repository and they manage by using repository in applications ever."* In general, this practice is benefits communication between developers because, in this way, everybody can learn from each other.

In coding standard, the programmers write all code in accordance with rules emphasizing communication through the code. Based on this, the researcher asked the participant several questions related on these issues. Such as, the Expert 1 said that, *"we as a team, always strive to adhere in the code standard"* Moreover, the participants also adds that, *"The programmers always worked efficiently in the*

presence of proper coding standards because it more systematic." This illustrates the adherence to coding standard among developer team.

With regard to continuous integration, the main aim of this practice is to integrate and build the system many times a day, every time a task in completed. This corresponds with what our participant mentioned, who referred that *"Frankly, I always try to synchronize and check my code each step from our progress."* Actually, any XP project is composed of a series of iterations that gradually evolves an artifact, this called Small release (or some authors called frequent release) practice. In this practice, the development team needs to release frequent iterative versions of the system to the customer. Our participant referred to this process through the interview session, for instance the Expert 1 said that *"There are three releases made before final releases"* This is critical in getting valuable feedback in time to have an impact on the system's development, this also stated by Expert 1, who also mentioned that *"Always the early reviews (release) will help in fixing better".*

Sustainable pace (also called 40-hour week), means the team have to maintain their productive in developing system by working only 40 hours per week. Through interview session, the participant (expert 1) confirmed that this practice fully applied by developers in UUM IT. The questions for this practice were focus on two issues: the rate of the hours work and the complete of the project. Our participant she answered on the first question as *"team who work in UUM IT and also the managers here, strive to do pace maintain work at the same hour rate.* "As well as, according to complete the project, she referred that *"The team always delivered a completed*

project at the exact time" However, the 40-hour week (Sustainable pace) is not a rule, sometimes it is more, and sometimes it is less. Thus, the important point is that the team must be fresh and creative.

Another XP practice is called pair programming. Briefly about meaning of this practice, all production code is written with two programmers at one machine. Based on this concept, several questions were asked for our participants to ensure applied this practice among the team. Many of the important tasks are not applicable related to this practice, such as, the Expert 1 said that: *"It is not easy to the team members accept pair programming"* In addition, pair programming is a collaborative approach that makes working in pairs rather than working in individual for code development. While, our participants stated that *"our developers prefer to work individually rather than as group while writing programs "*Also, another character for this practice, the two programmers switch their roles after some time. Where, pair programming seems to be dependent upon collocation. However, this feature not applied among the UUM IT team, and this confirmed by our participant (Expert 1), who stated that: *"The swap between partners always low."* And this asserts this practice not fully implement properly.

In fact, XP succeeds by making a project resilient. Therefore, in the test first programming practice, the programmers continually write unit tests, which must run flawlessly for development to continue. Meanwhile, the customers write tests demonstrating that features are finished. Based on our participant this practice also not implemented in UUM IT. Where the Expert 1 referred that *"The test cases developed always applied at the end of project"*As well as, she said that *"The time to*

test-first design really very low." The participant was unable to apply this practice during the certain project, but actually the UUM IT team applied it well at the end of the project.

Put simply, design and implement only what is needed today. Therefore, the system should be designed as simply as possible at any given moment. XP emphasizes keeping things as simple as possible, this also referred by our participant, who asserted, *"In the center we do the simplest thing that can possibly work, as well as we used the standard design language such as UML diagrams for better communication and also for clarify among the team and the managers."* A standard design methodology such as UML remains a design simplicity that all the team can understand.

Each practice on the XP depends on each other. Therefore, this qualitative study focused on all XP practice. As a result, it is important to understand the use of refactoring in XP Method. During the interview with the expert 1 mentioned for use this practice thru this statement *"The reuse the code always help to speed up the development process because easy for us no need to recode another code".*

In the last practice, both the customer and the programmers share a story based on a metaphor that guides all development by describing the functionality of the system. In addition, the team shares some common understanding from their past experiences. In this practice actually our participants are conflict about if they use it or not. For instance, the expert 1 stated that *"always classes and methods have good descriptive name, while other member in the certain team don't need often to ask and refer to understand the architecture"* Furthermore, Metaphor is achieved when

the members and client shared common vocabulary to communicate, our participate also added that, *"our client understand and explain that metaphor"* In sum up, based on this Expert 1, two of the XP practices are not fully implemented in the UUM IT development software team.

4.4.2 Expert 2

This study focused mainly on the expert's viewpoint who has experience with software development in UUM IT (formerly Computer Center). Twelve practices are the core our objective of the present study. For understanding how the UUM IT team harness these practices, five expertsparticipated in this empirical study. Expert 2 sheds light on several issues related to XP practices, some of these issues similar to the previous expert while some different. Figure below depicted the outcome from Nvivo based on the twelve practices.

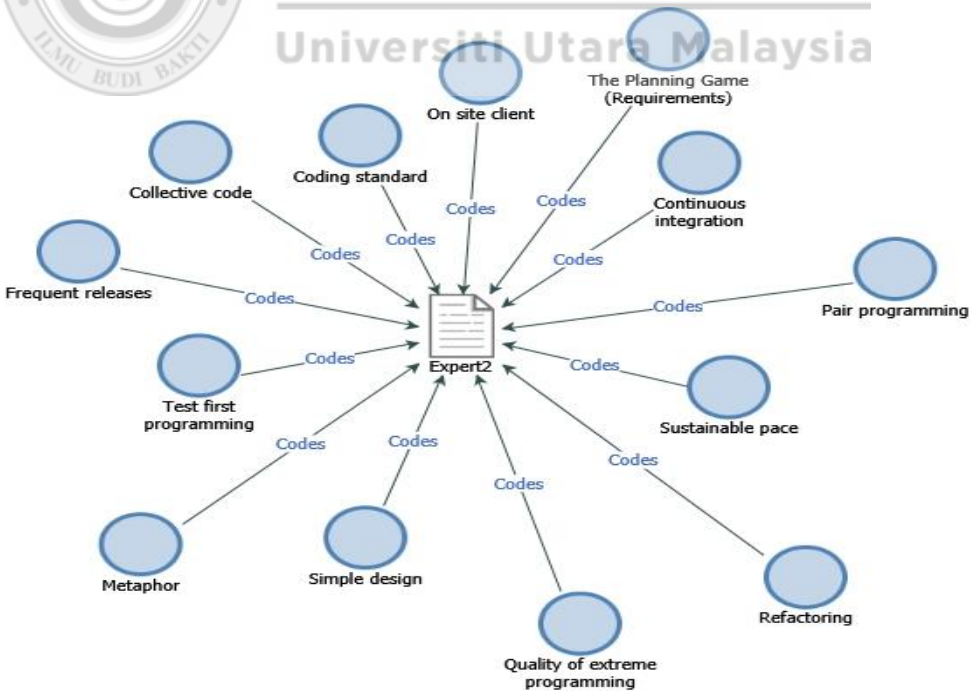

Figure 4.4 Expert 2 with XP Practices

The first set of the questions of the interview session was about the "On-site customer". The answer of our participant (expert 2) was quite similar to the former participant (Expert1) in respect of this practice. For instance, Expert 2 pointed the client is considered as an important part during development the software or the application. Where she stated that *"Actually, before started any project, we as the team strive to get the feedback from our client based on the requirements"* Furthermore, she also indicated to the manner to communicate with the client. As an example, she said that *"we use the traditional manner or the modern technologies to communicate with the client to get the feedback"* This illustrates this center is fully used this practice this results based on the Expert's perspective.

Another practice is Planning Game, or as mentioned earlier some authors also called "Requirements", this is another XP practice associated to our study. The expert 2 and based on several issues related to this practice stated that *"We always seek to write the user requirements"* As well as, this practice focus on the interaction among clients and the team member, and this confirmed by our participant, who referred that, *"I as the member of the UUM IT's team, always discus with our client about the certain requirements"* The expert also stated, *"The development team also estimates the time needed to complete the test based on the requirement"* and also the priority*" according to clients' need and urgency"* Therefore,based on expert's viewpoint, this practice is a quite adopted in the UUM IT.

With regard to collective code practice, two things relate to this practice, we strive to concentrate through the interview, change code and code repository. Regard to change code, the present participant stated *"Actually, in sometime the member of the*

team has the opportunity to change the code that s/he has not originally typed, but indeed this process, not always" Meanwhile, she also adds about the second issue (code repository), *"With each project, the UUM IT team has a code repository and they also manage the repository by using the repository in application server".* In fact, the key aim of this practice is each member of the team is responsible for all the code. Thus, means that everybody is allowed to change any part of the code. In turn, based on our participant (Expert2) this practice not fully adopted among the UUM team.

For a team to work effectively in pairs, and to share ownership of all the code, all the programmers need to write the code in the same way, with rules that make sure the code communicates clearly, this called coding standard. In coding standard questions, the answers of this Expert were quite similar to the previous participant. He stated that *"regarding to the coding standards is very important and as the developer I always follow the rules throughout the project"* However, this expert also mentioned to some difficulties sometime when work as the team to follow the certain rules through development some applications, where he said that *"in case work as the team, faces tracking the code, due to difference in coding standards between two programmers who working on the same project".* While, use the rules for naming and formatting code unites is very important to make the system more consistent so that it is easier to read and understand. Based on this interview, the UUM IT team adopted this practice partially.

The XP emphasis on continuous integration against a potential loss of configuration control, therefore our participant stated, integrate and build the system many times a

day, whenever a task is completed. Where, he said that, *"when I or my team finish each feature, we always strive to integrate this feature and check it"* this in fact, one of the XP Characteristics to keep the system fully integrated at all times. Comparing with previous participant (Expert 1), expert 2 stated that *"for this issue <code integration tool>, I'm particularly not use any tool for this purpose".* Continuous integration is deemed as a key practice of XP method, and must be introduced as soon as possible on every software development project. Building on results of this participant, not all the elements related to this practice are adopted among UUMIT team, therefore, partial adopted.

The small frequent release cycle help customer to gain confident in the progress of the project and enable the developer to tracking the progress. This also confirmed by our participant (expert 2) when he said that *"Always, the early release gave the team an opportunity to improve the project and also following the customer's requirements"* In addition, he stated *"there are three releases before the final ones"*according to our participant this practice fully adopted among the UUM IT team when developing any project.

XP advocates that programmers do not tire themselves out by overworking themselves. Based on the participant, no project appears to have suffered extended periods of long hours. He said that *"As a leader of some software projects, I seek to not put my team in overworking, because the team will become more creative if they are rested, and healthy"* Meanwhile, related to achieving the projects in the specific time, he said that *"We are always working to finish projects on time"* Therefore, this expert confirmed this practice is fully adopted.

In pair programming, two persons design, code and test software together at one computer. Related to these concepts, there are several issues was requested from our participant to answer it. With regard to switch, the switching did not happen between partners as the participant mentioned, who stated that *"really hard to swap between the partners"* As well as, if the team accept to work as pair, he said *"It is not easy to my team accept pair programming"* many issues he mentioned affected on use pair programming such as "*education*" and "*experience and personality of developers*" In fact, this practice not adopted among UUM IT team, according to expert 2.

Another practice of XP method discussed with our participant was tested first programming. It is a software development practice which has been proposed for decades. Many issues answered by the participant related to this practice. For instance, he stated *"the final product always testing after the complete"* In addition, he adds that, *"As a team, we testing the project, but can't call it this a test-first programming"* In fact, Testing is an essential part of XP.

In XP method, the system should be designed only for the current requirements and not for the future enhancement. Therefore, design must be simple, yet precisely aligned with the client requirements. This confirmed by Expert 2, who stated that *"Actually, we believe that simplicity is a very subjective idea, especially when there is a knowledge gap among developers."* In fact, the great importance to keep things as simple as possible clearly emerged with XP. As well as, he said that *"complex design is more difficult to understand than a simple design"* Meanwhile, also stated *"personally, I'm always trying to simplify the work to become understood to team*

members" the simple design practice adopted in UUM IT based on the expert's viewpoint.

In this practice (Refactoring), programmers restructure the system without changing its behavior to remove duplication, improve communication, simplicity, or add flexibility. Specifically, XP team tries to reuse a coding as much as possible. Our participant mentioned that *"reuse of coding useful in our work, and actually we are using this technique"* through the interview session. This proves the UUM IT team adopted this practice during designing applications.

With regard to the last XP practice, namely Metaphor, it describes the overall shape of the system created by mutual understanding between users and developers. The expert's answer was more pronounced, where he said that: *"Honestly, we don't know this practice, and based on my knowledge we don't use it"* Consequently, based on the expert this practice still not adopted among UUM IT team.

4.4.3 Expert 3

The key goal of this study is to understand, if UUMIT team adopted all or most of the XP practice. Therefore, this study based on the experts' experiences who work in UUM IT. We asked a series of questions to our participant related to this phenomenon. In the following the answers of the expert about twelve XP practice. The first practice in our study is, On-site customer. The expert 3 referred to the issues pertaining to this practice as *"We seek to get the feedback from our clients as a weekly"* In addition, she also stated the manner of communicating with clients to get the feedback, such as "*In the center, we can get the feedback from the client by*

several ways, for instance face-to-face or by email or thru instant chatting" Figure below shown the XP practices based on the expert 3. In the same practice, the expert adds that, *"after get the feedback from the certain client, the client not allow for them always change the requirements".*

Regarding to Planning game practice, our participant confirmed she adopted this practice. For instance, she said that *"Before all, I am always writing the user requirements"* and also *"discuss the requirements with our clients*" based on the expert's answers this practice fully adopted. Whereas, collective code ownership practice has not been applied in full among the UUM IT team as mentioned by expert 3, where she said that *"Frankly, it is difficult for my team change certain code, when they actually not the original writer"* In the same vein,she stated about code repository, where said *"My team have a code repository"* .

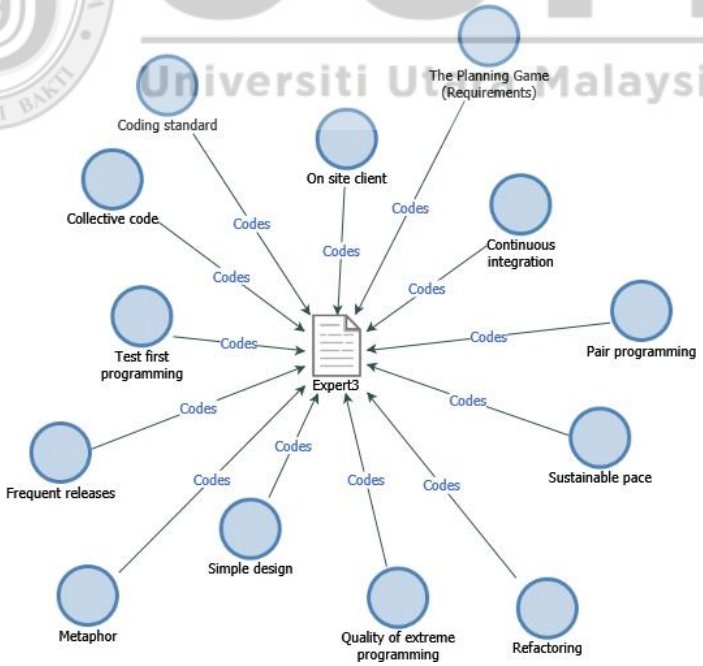

Figure 4.5 Expert 3 with XP practices

With regard to Coding standard, she agreed what the previous experts mentioned and she stated that *"In reality, not all the teams follows the same code standards"* with respect to Continuous integration, her answer was quite similar to the expert 1, where she confirmed that, *"thru the development process, I always check the code"* and also *"I use source code control to assess code integration"*. As well as, based on the experience of the expert 3, the frequent releases practice adopted. Where, she indicated that *"the first release in weeks after commencing the project*" Thus, this admitted, confirmed there are several releases before the final one.

Additionally, during the interview session, our expert stated some information related to sustainable pace. She said that *"My team implement the requirements in the fixed time without overworking"* and *"always my team delivered a complete project, and not need to work overtime"* This emphasizes the UUMIT adopted this practice. In the same context, among all of the XP practices, one of the key practices is pair programming. Pair programming is not just one person programming and the other observing. Instead, it is a dialog between people trying to simultaneously design, program, analyze, test, and understand together how to program better. The expert said *"It is not easy for the team members accept pair programming, because we do not have enough staff for pair programming and all the developers set together in the same room, thus if any member need help can ask direct"* but actually this not achieve the pair programming concept. Therefore, we can assume that, pair programming not adopted in UUM IT.

Test first programming practice also discussed with our participant (Expert 3). During the interview, the expert mentioned for many issues confirmed they not

adopted this practice. Such as, she said *"We always testing only the final product"* Test-first programming was difficult to implement at first. In fact, the participant answer (expert3) was quite similar to the previous ones. In contrast, simple design practice, she stated that, *"To make the system or the process simple, very important for the end-user and for partners, therefore I attempt to simplify things during the work"* and thus attained the objective of this practice, is the simplest possible design for implementation at the present moment.

Improving the code structure while preserving its function. Reuse code, removing duplications, improve communication and make the code flexible all these under refactoring practice. In this issue, our participant indicated that *"Actually, reuse the existing code < such as methods or classes> useful and reduces the time"* and she add that *"Of course, we are using the former code from different systems".* Turning to the last XP practice, namely Metaphor, it's a way or tools for description of developer and client of how the system will work. Based on the expert's experience, she confirmed that *"We do not use this concept in UUM IT currently"*

4.4.4 Expert 4

XP covers most of the software development life cycle. Therefore, become more important to understand, if UUM IT center adopted all its practices. Best way to attain this goal, through carry out the interview with people who have good experiences with this phenomenon. In the previous sections, we highlight the experiences of three experts with XP practices, currently, we seek to analyze the fourth expert interview.

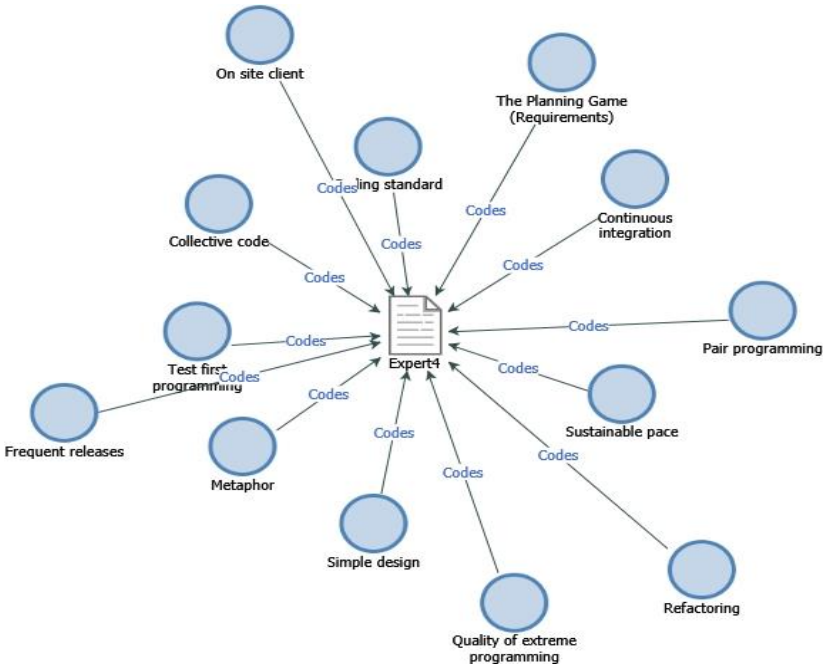

Figure 4.6 Expert 4 with XP practices

He asserted on the importance of the client's feedback, where he said that *"we always strive to get feedback from our client whether daily or weekly"* as well as, he mentioned on the manner to get the feedback from the client, he stated *"I get feedback from our clients through using phone"* In fact, all these issues that referred by participant, under the on-site customer practice. Planning game, this practice improves the communicative between the developers and clients. Thus, several questions related to this practice have been developed. The expert said that *"Actually, before commencing any project, we attempt to write the user requirements"* and with regard of discussion with client, he indicated that *"discussion with client frequently about their requirement will improve the system and enhance the final product".*

Collective code ownership, we also discussed it with our expert, to confirm if they adopted this practice or not. He stated that *"Honestly, very difficult for any member change the good he or she not the person who wrote the original code"* while, the main purpose of this practice is encourages everyone to contribute new ideas to all segments of the project. However, there is another feature for this practice applied on the UUM IT, where the participant said *"My team has a code repository…"* Also we debated with an expert about coding standard. In fact, they face challenges to fully adopt this practice, he said *"each team has its own style in describing the code"* and also stated *"Actually, this difference,make difficult to deal with coding".*

With respect to continuous integration, he stated that *"Every now and then, during period design any project, we check the code"* this practice also linked with another XP practice, called frequent releases. The expert said about this practice that *"the first release in weeks after start the project"* Thus, several releases following the first ones. Quite similar to the previous experts, this participant also mentioned to the sustainable pace, where he said that *"In the work, the developer seek to keep a normal work schedule to remain productive and interested in the project"* Also confirmed what the previous experts said, he said *"Our team always delivered a complete project"* In contrast, the pair programming practice was difficult adopted by UUM IT team. *"Is not easy for the team members accept this concept, maybe in the present time"* the expert said. In the same context, he also stated the Test first programming. *"For me and my team, we always check and test the whole project in the end"* as he refereed.

Another two XP practices (Simple design and Refactoring), our expert mentioned, they are used these concepts in their work. With respect to simple design practice, it can facilitate communication between developers and project managers. Our participant (Expert 4) referred that *"We attempt to simplify our process to become clearer for others"* and *"always success when we keeping things simple"* Meanwhile, he said that *"we can reuse any code, when we needed"* this was related to refectory.However, still metaphor partially among UUM IT team, based on our expert. He said clearly that *"Frankly, our clients understand and explain metaphor"*.

4.4.5 Expert 5

For focus in depth, five experts were selected for the current study. Five experts have a good experience with present phenomena. The last interview was conducted with Expert 5, who has twelve years experiences with software development and UUM IT (formerly computer center).

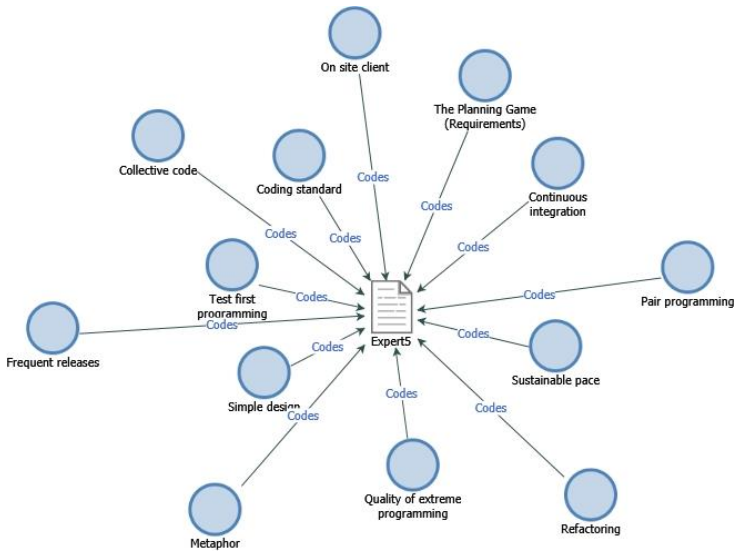

Figure 4.7 Expert 5 with XP Practices

In fact, after interviewing with the Expert 5, we reveal that, there is quite similarity of some answers with the previous participants. As the former experts, we will highlight his answer practice-by-practice. With regard to on-site customer practice, it ensures that the developers stay focused on the requirements. This agreed with our participants' answer, where she said that, *"Our clients always help thru give us clear requirements"* " In addition, if developers lose focus, the client is there to help the developers regain focus in order to satisfy the project requirements. Where, the Expert said that *"I get feedback from the clients through the traditional ways < such as face-to-face> or by email"* In the same vein,she stated about a planning game that *"I am always writing the user requirements"* Also, *"I am discussing the requirements with the client"* therefore, based on the answers of this expert on these two practices, they are adopted it.

Turning to collective code ownership, it means every developer has ownership of all development documents and program code, and can make modifications anywhere and at any time. In fact, this process, not fully adopt among team of UUM IT as mentioned by expert 5. Who said that *"In fact, sometimes can any member of the team change the code that they not write it"* likewise, the coding standard practice, he said that *"Not all the development team follows the same code standard"* the interviewee also agree that *"will become good to have a common code standard"* also *"the first release cover most of the functionalities".*

With respect to continuous integration *"Actually, this process importance of each project, we therefore always check the cod*e" As well as, *"We used VCS tool to assist source code integration"* This analysis is similar to expert 1, 3 and 4. In contrast,

frequent releases practice, the expert agreed with all previous interviewees. She stated that *"In fact, weeks after we start the project, the first released"*. Generally speaking XP works to the times of a normal working week and thus limits the amount of overtime a team will undertake. This supported what our expert said about Sustainable pace, who stated that *"there was not a change in the number of hours the developers worked per week"* As for thePair programming and Test first programming practices, our participant (Expert 5) asserted these practice not adopted among team of UUM IT. Where, she said that *"sometimes, the member works alone about <methods or interface> therefore really difficult sit long time with other team member just for discussion",* while about the test first programming practice, she stated that *"test first design not applicable"*.

As for another two practices the Simple design and Refactoring. The participant (Expert 5) confirmed on simply design quicker to code and easier to maintain than a non-simple design. She said that *"any system will succeed just when design in a simple manner"* Likewise, she asserted that *"we reuse the former code, without change in the substance"* on the contrary, she confirmed "we are not discuss every time with the client until complete whole project" and *"Honestly, I don't use this concept <metaphor> in my work"* This expert extremely agree with previous experts who conducted with them interview about the metaphor practice.

4.5 Discussing of Findings

This section discussed the results after analyzing in the chapter four. In fact, the discussion depends on the practices. Based on the answers from the experts about first practice ***(On-site customer)*** were referred to how many times that the team

spends with the client to set immediate and continuous feedback when developing software. According to Koskela and Abrahamsson (2004), the most interesting XP practice is the on-site customer. The experts confirmed that the UUM IT team gets feedback from client by email, face-to-face, meeting, or also by phone. This also supported by Kircher (2001) and Kircher and Levine (2000) and Kircher, Jain, Corsaro and Levine (2001), who referred adopted instruments such as email to get the feedback from customer to developer or among the team can increase the communication. Where, lack of sufficient communication between people can lead to serious problems in a project. Meanwhile, According to Kircher *et al*, (2001) one of the key requirements of Extreme Programming (XP) is strong and effective communication between the team members.

Furthermore, the results indicated that the client takes all the time when have a meeting with the developer, as well as, the client helpful to give clear requirements. Our experts also confirmed that, they get clear requirements mainly from the client, surfing internet, and ask opinion from expert domain. With regard to times for feedback, the results also highlighted that, the number of times to get feedback from the client daily or weekly and based on the requirements and also the client always give immediate and consistent feedback for the developer. This period to get feedback also stated by Kircher *et al*, (2001) who pointed that, daily or weekly reports with feedback from customer to developer can also increase the communication, as well as the familiarity, that is the spirit of collaboration and trust between the stakeholders. Thus, on-site customer practice aimed at encouraging

communication channels to remain open at all times. In fact, on-site customer is fully applicable among the stockholders in UUM IT.

The second practice focus on the present study was the ***planning game***. Based on the interviews the results indicated that, this practice adopted among the UUM IT team. In fact, this XP practice concentrates on the requirements and discusses about it among clients and developers. The team of the UUM IT discussed the requirements with the clients before any project related to the client. According to Tian (2009) in the planning game, customers decide the scope and timing (requirements) of the release based on estimates provided by the developers. In the same vein, Macholz (2007) stated that, the planning game is a series of activities between the programmers and the clients that define what features will be implemented in the next iteration of the software. Thus, the outcome proved that the UUM IT used this practice when developing the software.

Twelve XP practices is very important for any software development institute, whether was private or public. One of the important practices is the ***collective code ownership***.This practice means that any one of the UUM IT team can improves any code anywhere in the system at any time if they see the opportunity. The results pointed out that, not all the developers of the UUM IT team have the opportunity to change the code anytime s/he need. However, the results also indicated that, the UUM IT team has server repository to save their code. Thereby, this practice is applied, but not fully adopted. In fact, collective code ownership is necessary for software development project. Whereby, Dudziak (1999) asserted that, if there is a need to edit some code in order to make the changes or to integrate the developer

must be able to do so. Meanwhile, also Gittins, Hope and Williams (2001) stated that, collective code ownership has many merits, such as, it prevents complex code entering the system, and developed from the practice that anyone can look at code and simplify it.

According to Singhal and Banati (2014) during code development, certain standards must be followed by the entire team as it keeps code simple and understandable by all team members. Therefore, with regard to ***coding standards*** the findings indicated that, UUM IT development teams are committed to the certain coding standards and follows the rules thru their work. However, the results also show that, there are several of the coding standards are applied in UUM IT team, such as *"team standards"* and *"software development standards"* At the same time, the results based on the expert's perspective stated that, there is a difference of the coding standards between two programmers partners working on the certain project. Although, Al-Tarawneh (2013) referred, the code should be clear to everybody in the project, in order that all the team members can make changes to it. Consequently, the outcome of the coding standards confirmed that this practice is used by UUM IT team, but need to become more consistent.

Another important practice of XP method is called ***continuous integration***. In fact, it becoming increasingly common in industry to code, tests, and integrate at the same time. According to the results of the previous chapter, this practice is done on an instant basis after developing a number of user stories. Where, implemented requirements are integrated and tested to verify them. According to Stamelos (2007) this practice is important for quality. As well as, the findings indicated that, several

tools used among the UUMIT team for this purpose, such as *"source code control"* and *"VCS tool".* However, some teams not use any tools. While, static analysis tools could also be applied when doing Continuous Integration (Fowler & Foemmel, 2006). As well as, Ambu and Gianneschi (2003) pointed out that, no doubt, continuous integration is a key practice of XP. Consequently, the continuous integration practice is adopted in UUM IT.

Addition to XP practice that earlier discussed, the small ***releases (frequent release)*** also help to conduct the consistency between the requirements and final project. The results indicated that, there are more than two releases made before the final releases. This also supported by Maurer and Martel (2002), who referred that, a short release cycle also helps developers deal with changing requirements and reduces the impact of planning errors. The findings also asserted that, the first release covers most of the functionalities in the specification document, furthermore, the early release helped in fixing bugs better. According to Al-Tarawneh (2013), the idea behind small releases is to get the system in production on time in order to get constant feedback from the customer, as well as to avoid risks, and minimize effort necessary to change the effect, this is exactly what the UUM IT team practiced. Therefore, this proves the UUM IT is completely adopted this practice.

This practice indicates that the software developers should not work more than ***40 hour weeks.*** The findings from five experts indicated that, they finish any duty or project without need the overtime or overworking. In addition to that they also stated that good health and rested will make a creative team. This also agreed with Zuiderveld (2003). Who referred that, programmers (and people in general) perform

their best work when they are well rested, upbeat, and healthy. As evident in other values expressed in the XP community, people play an important role, and are not considered simply robotic programmers. Therefore, this confirmed this practice was adopted among UUM IT team.

As for ***pair programming***, refers to two programmers can work together as a pair on the workstation/computer, one is the driver (write code) while the other observer assist the driver and suggest a solution to the driver. The findings indicated that, this practice not applicable among UUM IT team. The experts highlighted into several issues prevent to adopt this pair programming, such as, education, experience and the personality of the programmers. While, Vanhanen and Korpi (2007) asserted, everyone should use pair programming for all development tasks from the start to the end. As well as, pair programming with pair rotation, help increasing the knowledge level of the individuals and subsequently of the team. The UUM IT center must strive to adopt this practice among the team, especially among new staff. Because pair programming, encourage the tacit transmission of knowledge and promote continuous training (Sfetsos & Stamelos, 2007).

XP covers most of software development life cycle. Therefore, the ***test first programming*** practice is important, because it's providing rapid feedback between customers and developers. The findings indicate that, some applications were tested only in the end of the project. While, the testing is deem as a one of the major building blocks of XP (Dudziak, 1999). As well as, Wood and Kleb (2003) stated that, comprehensive test coverage is the key to XP method. The repeated tests will assist to ensure that, the system remains intact after changes and that it moves in the

direction the customer wants it to. In fact, the findings also indicate that, there is only programmer testing. While, Al-Tarawneh (2013) pointed out that, the programmers write the unit tests and the customer writes the functional tests.

This practice facilitates communication within developers, and between developers and project manager. The findings shown that, the ***simple design*** practice adopted among the team of the UUM IT. They use a standard design language such as UML diagrams for better communication in the project. This supported by, Al-Tarawneh (2013), who stated that, we suggested to use UML to simple design and improve the communication between the developers from different cultures.

XP programmers improve the design of the software through every stage of development instead of waiting until the end of the development and going back to correct flaws. Based on the answer from experts the team always reuses the code if they need the code. This also stated by, Macholz (2007) who said that XP teams work toward the "once and only once" principle to coding, and try to reuse as much as possible. In addition, the experts confirmed that always the reuse code can help to speed up the development process because not need to rewrite another code and they still use the code if can use it. Thus, the result shows that, this practice is adopted among UUM IT team. According to Cao, Mohan, Xu and Ramesh (2004) ***refactoring*** is a way to improve the design and making the system more robust.

While, ***Metaphor*** used by the programmers to help communicate ideas and explain concept to customers. In general, it provides easy understandable communication platform for developers, project manager and customers. The outcome of interviews indicates that, most of the experts did not know this practice and they do not use it,

this agreed with Beck (2002), who noted that people have difficulty understanding Metaphor. In contrast, two of our experts, they sometimes use the metaphor. Metaphors can provide common vision and feeling to the system (Macholz, 2007). As well as, he metaphor can be a useful tool to aid in communication among team members (Siebra, Mozart Filho, Silva & Santos, 2008), especially new staff in this center. Table below summaries the final finding based on the experts' experiences. Eventually, this study indicated the most XP practices use among UUM IT teams and those use whether partially or not use, the Table 4.3 depicts summarized the experts findings. In fact, determine the degree of the used the XP practices (applied, partially and not applied) based on definition of each practice and compare with experts' interview. In addition, these findings degree also confirmed by the experts, see the Appendix B.

Table 4.3 *Summaries the final XP practices based on the experts*

XP PRACTICES	**EXPERT 1**	**EXPERT 2**	**EXPERT 3**	**EXPERT 4**	**EXPERT 5**
On-site client	Applied	Applied	Applied	Applied	Applied
Planning game	Applied	Applied	Applied	Applied	Applied
Collective code ownership	partially	partially	partially	partially	partially
Coding standard	partially	partially	partially	partially	partially
Continuous integration	Applied	Applied	Applied	Applied	Applied
Frequent releases	Applied	Applied	Applied	Applied	Applied
Sustainable pace	Applied	Applied	Applied	Applied	Applied
Pair programming	Not applied	Not applied	Not applied	Not applied	Not applied
Test first programming	Not applied	Not applied	Not applied	Not applied	Not applied
Simple design	Applied	Applied	Applied	Applied	Applied
Refactoring	Applied	Applied	Applied	Applied	Applied

Metaphor	Applied	Not applied	Not applied	Applied	Not applied

4.6 The XP Quality Implementation

As for quality implementation, Expert 1 mentioned in the important of the interaction between the customers and the developers through the project implementation, and this can effect on the quality of the final product. And she said that *"The communication between the customers and the developers need to focus more to improve the final applications"* While, the expert 4 concentrates of the coding standard as the important practice to enhance the quality implementation. Thus, he referred that *"When have the same coding standard this will effect positively on the quality of the software"* While, the rest of the experts stressed the importance of all the XP practices to improve the quality of programs. According to Sfetsos and Stamelos (2007) the XP practices are aimed at improvement of quality, this also supported by Xu (2009). Where, XP's practices focus on improving communication between among all project stakeholders (Developers, customers, and project manager).Therefore, Beck (2000) emphasizes the importance of using every practice, "*Any one practice doesn't stand well on its own. They require other practices to keep them in balance*".

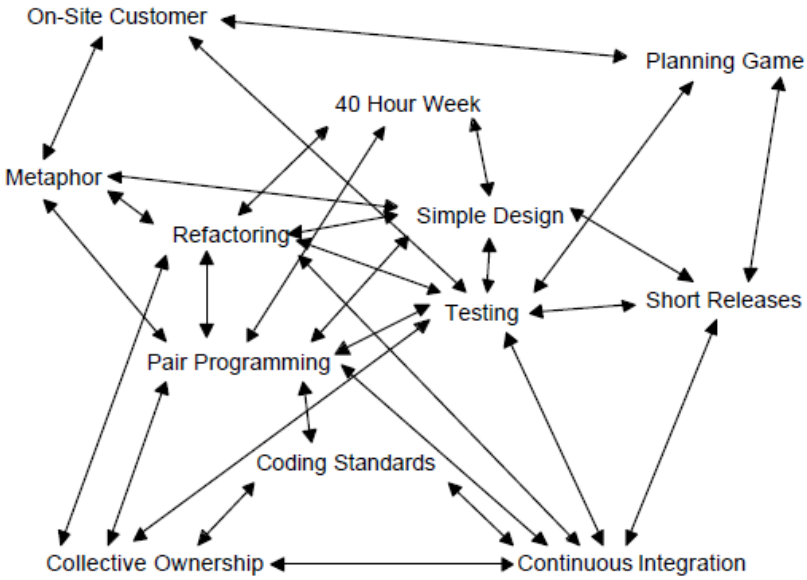

Figure 4.8 X Links between practices

However, through conducting the interview with five experts from UUM IT, the findings revealed that, two out of twelve of the XP practices not adopted. Furthermore, three are partially adopted, whereas, the rest of these practices were adopted. The UUM IT center should adopt all the practices one time, because the shortcomings of the individual practices are compensated by the strengths of the others (Skinner & CIS, 2001; Alshehri, 2014; Stellman & Greene; 2014). As well as, XP is considered best practice to improve the software quality by repeated feedback and changing requirements. By and large, in Table 4.4 below summary the experts' perspective related to XP quality implementation.

Table 4.4 *Summary of the XP quality implementation findings based on the Expert's opinion*

XP practices	Degree adoption	Themes	Notes
On-site customer	Applied	(get feedback from the customer),(daily or weekly) (clear requirements)	The results based on the experts' experience indicated that, they applied this practice through to get feedback from the customers sometime daily or weekly. Actually, by using this practice can increase the communication between developer and customer, this will help to set clear requirements for the certain project. Furthermore, they used various manners to get these requirements from the clients.
Planning game	Applied	(write the requirements),(discuss the requirements with customers)	The results also indicated that, the developers discuss the requirements with customers. In fact, based on literature this process will enhance the collaboration between clients and programmers by discussing and understanding the requirements between them. Thus improve the software quality.
Collective code ownership	Partially	(change the code), (code repository)	The former studies stated that, practice enhance the communicative among the developers, by using this practice the developers can learn from each other. However, building on the findings, not all the teams on the UUM IT harness this practice through building any software project. Where through

			developing any software project sometime the member not have the right to change the code for any programme under writing. In the same vein, they referred to the code repository.
Coding standard	Partially	(code consistent), (easy for use)	The main purpose of this practice is to keeps the code consistent and easy for the whole the team to read. Unfortunately, the findings indicated that, there are variations in use this concept among UUM IT teams. The development of the software is done iteratively and phases sometimes overlap, therefore the neglected or omit some practice maybe affecting the implementation the system.
Continuous integration	Applied	(integrate the code) and (automated tests)	In fact, this practice enables the integration of the changes to the code very often. Results were much identical of this concept. Where, the team members always integrate the code and automated tests to ensure the whole code is working. Thus, provides developers with rapid feedback on the quality of the code.
Frequent releases (small release)	Applied	(several releases) and (Frequent releases)	The findings have indicated that, there are several releases before the final ones. Actually, in this case will rapid feedback between developer and customers where this has an impact on the system's development. In addition, enhance the sharing

			knowledge between developers.
Sustainable pace	Applied	(overwork), (same hour rate), (complete project)	When there is a fatigue in the work the developers may commit more mistakes. Based on the findings, no project appears to have suffered extended periods of long hours. This will help the developer to deliver complete project with the availability of suitable work environment.
Pair programming	Not applied	(communication between the programmers), (sharing knowledge)	The goal of this practice is to give high quality code and enhance the relationship and communication between the programmers to learn from each other and also sharing knowledge between them. The finding have shown that, number of factor effect to harness this practice among UUM IT team, such as education, experience, or also the personality of the developers. Not apply this practice actually will effect on the quality of code and also the communication between the programmers, especially the new staff or programmers.
Test first programming	Not applied	(tested only in the end), (the client not part of it)	It is important to get fast feedback between customer and developer and also give good quality code. Actually, this different from the requirements feedback. However, the findings indicate that, some applications were tested only in the end of the

			project. Furthermore, this testing the customer not part of it.
Simple design	Applied	(UML diagrams)	Based on the findings, UUM IT team use a standard design language such as UML diagrams for better communication in the project. And this supported by previous studies, who stated that, we suggested to use UML to simple design and improve the communication between the developers from different cultures.
Refactoring	Applied	(reuse the code)	The findings indicate that, the team strives to reuse the code if they need the code. This also stated by, Macholz (2007) who said that XP teams work toward the "once and only once" principle to coding, and try to reuse as much as possible.
			Important to improve the communication between the customer and developer and it is easy way to explain how the system works. This practice linkage with the previous practice (test-first programme). The client is act as a key element in this practice.
Metaphor	Partially	(Project teams explains to clients), (communication with clients thru the project)	Based on the findings, some of UUM IT team don't have the times to explain and discuss with client and explain the entire project. In contrast, have some teams strive to explain for clients.

4.7 Summary of Chapter Four

The main purpose of this chapter is to achieve the objectives of the current study. The literature review and interviews are essential resources for this purpose. In the two sections, highlighted and illustrated the XP practice based on the previous studies. While, in the third section, analyzed the interviews to reveal if the UUM IT team adopt all the XP practices or not. The findings indicated that, more than half of these practices were adopted by the team of UUM IT.

In fact, most of XP practices, such as pair programming, encourage the tacit transmission of knowledge and promote continuous training. Therefore, in this study, we recommended the UUM IT teams to adopt this practice, to help of increase the knowledge level of the individuals (especially the new staffs) and subsequently of the team. Meanwhile, to increase the code quality and reduce defects through the implementation the software. Therefore, this study also recommends that, the UUM IT team should fully adopt the collective code ownership practice in pair to get higher quality of code and also increase the trust between the team members. Also, the constant interaction among the clients and the developers in the UUM IT center is essential to enhance the software quality. For this purpose, the team of the UUM IT should fully adopt the Metaphor practice and explain what mean for each team. Thus, this will help to increase the communication between them and also to find the right place to put the functionality. Furthermore, we also recommend adopting the test first programming increase the quality of code productivity, where, when the team harnesses this practice will helps detect errors in the code before delivered to the clients. As well as, the XP practice works together and also facilitate transfer the

knowledge among the team member, therefore must there consistence when used the certain roles or code. Thus, this center should adopt all the characteristics of the coding standard practice. In fact, to improve the software quality all of the XP practice must be adopted.

CHAPTER FIVE
CONCLUSION

5.1 Introduction

This chapter discusses the outcomes of the study based on the results of the semi-structured interviews that were conducted in the UUM IT to assess the quality of Extreme programming implementation. The research objectives were achieved in the first section, which discusses each of the objectives of the current study. In the second section, the limitation of the study was identified. The future work was done in the third section, and the contribution was highlighted in the fourth section.

5.2 Achievement of Research Objectives

The current study aims to investigate the quality of Extreme programming practices used among UUM IT developers. In order to achieve this main objective, three sub-objectives have been identified. Hence, this section attempts to discuss the results that support the objectives of the current study.

5.2.1 Objective One

The first objective of this study is to identify the best practices for evaluating the quality of Extreme programming (XP) implementation. XP has been chosen in this study 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. These practices are: On-site customer, Planning Game, Collective Code Ownership, Coding standard, Continuous

Integration, Frequent Releases, Sustainable Pace, Pair Programming, Test First Programming, Simple Design, Refactoring, and Metaphor. Many researchers state that these practices must be used together and support each other to get high quality XP implementation. This objective was achieved by reviewing the previous work, and detailed of explanation of each of the practices has been discussed in chapter two (see Section 2.4).

5.2.2 Objective Two

The second objective is to propose the conceptual model for evaluating the quality of Extreme programming (XP) implementation. Based on the first objective, this study has proposed a conceptual model for evaluating the XP implementation. The proposed model was used to set the information about the XP practices.

5.2.3 Objective Three

The third objective is to conduct a case study at UUM IT based on the proposed model. This study used qualitative approach to get more in depth information, and UUM IT was chosen as a case study. The current study chose five experts who work in the UUM IT and who have more than ten years experience in software development. This case study was conducted by using semi-structured interviews to get more details about the practices used by the UUM IT developers that help to achieve XP quality implementation.

In general, based on the experts' experience, this study has identified seven practices fully applied by the UUM IT team, which are: On-site customer, Planning Game,

Frequent Releases, Sustainable Pace, Continuous Integration, Simple Design, and Refactoring. Furthermore, the interviews also showed there are three XP practices used by software developers in UUM IT that need to be improved; these are: Collective Code Ownership, Coding standard, and Metaphor. The results also showed that there are two practices not applied by the UUM IT team: Pair Programming and Test first Programming.

5.3 Contributions of the Study

This study contributes significantly to the software engineering (SE) body of knowledge, specifically in the quality implementation of XP practices. Hence, more clarity has been made to understand deeply the importance of Extreme programming (XP) in software development. Particularly, a conceptual model is made to give a clear understanding on the best XP practices. Furthermore, this study showed how qualitative approach can aid to get in-depth analysis of the Extreme programming (XP) practices.

Meanwhile, a case study of UUM IT has been conducted to provide some qualitative evidence on the Extreme programming implementation in UUM IT. This is important because it will get more information to improve the practices in the organization.

5.4 Limitations and Future Work Directions

This study carried out only one case study in one computer center in Malaysia; this center is in University Utara Malaysia under the name UUM IT. This study also focused on one agile method called Extreme programming, where twelve practices

of XP method were used to evaluate the XP quality implementation t used among UUM IT developers. As mentioned earlier, the sample size of the study is five experts to obtain more in-depth information.

Therefore, in the future research directions, more than one case studies (Universities) can be carried out and the qualitative approach can be employed to generalize the findings. The next study can also focus on the other agile methodologies such as Scrum used in other centers to highlight which method delivers good quality services to the employees and end-users.

Furthermore, a comparison study can be carried out using agile vs. non-agile methodology to show the effectiveness of methodology used and the best practices for organization improvement can be proposed.

REFERENCES

Abdullah, M. S., al-Tarawnehb, M. Y., & Alia, A. B. M. (2012). Software process improvement in small software development firms. *Computer Science, 1*, 782-787.

Abrahamsson, P. (2003). *Extreme programming: first results from a controlled case study.* Paper presented at the Euromicro Conference, 2003. Proceedings. 29th.

Abrahamsson, P., Conboy, K., & Wang, X. (2009). "Lots done, more to do": the current state of agile systems development research.

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. Agile Software Development Methods: Review and Analysis. 2002. VTT Publications: Finland.

Abrahamsson, P., Warsta, J., Siponen, M. T., & Ronkainen, J. (2003). New directions on agile methods: a comparative analysis. Paper presented at the Software Engineering, 2003. Proceedings. 25th International Conference on.

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.

Ackroyd, S., & Hughes, J. A. (1992). *Data collection in context*: Longman London.

Agarwal, N., & Deep, P. (2014). *Obtaining better software product by using test first programming technique.* Paper presented at the, 2014 5th International Conference Confluence The Next Generation Information Technology Summit(Confluence).

Aguanno, K. (2004). *101 Ways to Reward Team Members for $20 (or Less!)*: Multi-Media Publications Inc.

Ahlemann, F., El Arbi, F., Kaiser, M. G., & Heck, A. (2013). A process framework for theoretically grounded prescriptive research in the project management field. *International Journal of Project Management, 31*(1), 43-56.

Alite, B., & Spasibenko, N. (2008). Project Suitability for Agile methodologies. *Umeå School of Business*.

Alshehri, S. A. J. (2014). *AHP-Based Methodology for a Complex Decision Support in Extreme Programming.* Faculty of Graduate Studies and Research, University of Regina.

Al-Tarawneh, M. Y. (2013). *Harmonizing CMMI-DEV 1.2 and XP Method to Improve The Software Development Processes in Small Software Development Firms.* Universiti Utara Malaysia.

Ambu, W., & Gianneschi, F. (2003). Extreme programming at work *Extreme Programming and Agile Processes in Software Engineering* (pp. 347-350): Springer.

Asnawi, A. L., Gravell, A. M., & Wills, G. B. (2012). *Emergence of agile methods: perceptions from software practitioners in Malaysia.* Paper presented at the AGILE India (AGILE INDIA), 2012.

Asnawi, A. L., Gravell, A. M., & Wills, G. B. (2014). *Significant aspects in relation to Agile usage: Malaysian perspective.* Paper presented at the Information and Communication Technology (ICoICT), 2014 2nd International Conference on.

Aveling, B. (2004). XP lite considered harmful? Extreme Programming and Agile Processes in Software Engineering (pp. 94-103): Springer.

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.

Baker, S. E., & Edwards, R. (2012). How many qualitative interviews is enough.

Beck, K. (1999). Embracing change with extreme programming. Computer, 32(10), 70-77.

Beck, K. (2000). *Extreme programming explained: embrace change*: Addison-Wesley Professional.

Beck, K. (2002). The metaphor metaphor. *Keynote speech-ACM OOPSLA, 2*.

Becker, C. H. (2010). Using eXtreme Programming in a Student Environment.

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.

Bird, M. (2007). *Comprehensive Examination Written Responses Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy.* Capella University.

Blokdijk, A. (2014). *Planning and design of information systems*: Academic Press.

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.

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.

Burman, E. (2015). Agile in action: Hybrid methodologies in practice.

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

Cagle West, M. (2010). Effective Software Engineering Leadership for Development Programs. *ProQuest LLC*.

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.

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.

Cao, L., Mohan, K., Xu, P., & Ramesh, B. (2009). A framework for adapting agile development methodologies. *European Journal of Information Systems, 18*(4), 332-343.

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.

Chung, L., Nixon, B. A., Yu, E., & Mylopoulos, J. (2012). *Non-functional requirements in software engineering* (Vol. 5): Springer Science & Business Media.

Cockburn, A., & Highsmith, J. (2001). Agile software development: The people factor. *Computer*(11), 131-133.

Cockburn, A., 2007. Agile Software Development: A Cooperative Game. 2nd Edn., Addison Wesley, ISBN: 0-321-48275-1, pp: 504.

Cohn, M. (2005). *Agile estimating and planning*: Pearson Education.

Conboy, K., & Fitzgerald, B. (2010). Method and developer characteristics for effective agile method tailoring: A study of XP expert opinion. *ACM Transactions on Software Engineering and Methodology (TOSEM), 20*(1), 2.

Creswell, J. (2009). *Research design: Qualitative, quantitative, and mixed methods approaches*: SAGE Publications, Incorporated.

Creswell, J. W. (2005). *Educational Research: Planning, Conducting and Evaluating Qualitative and Quantitative Research.* New Jersey. Pearson.

Creswell, J. W. (2006). *Educational Research: Planning, Conducting and Evaluating Qualitative and Quantitative Research*. New Jersey. Pearson.

Creswell, J. W. (2012). Qualitative inquiry and research design: Choosing among five approaches: Sage.

Creswell, J. W. (2013). Research design: Qualitative, quantitative, and mixed methods approaches: Sage publications.

Creswell, J. W., & Clark, V. L. P. (2007). Designing and conducting mixed methods research.

Cyganek, B., & Siebert, J. P. (2011). *An introduction to 3D computer vision techniques and algorithms*: John Wiley & Sons.

Da Silva Estácio, B. J., & Prikladnicki, R. (2014). *A Set of Practices for Distributed Pair Programming.* Paper presented at the ICEIS (2).

Da Silva Estácio, B. J., & Prikladnicki, R. (2015). Distributed Pair Programming: A Systematic Literature Review. *Information and Software Technology*, *63*, 1-10.

Darwish, N. R. (2011). Improving the Quality of Applying eXtreme Programming (XP) Approach. *The International Journal of Computer Science and Information Security, 9*(11), 16.

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).

Dey, I. (1993). *Qualitative Data Analysis: A User Friendly Guide for Social Scientists*, London, Routledge.

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.

Douglas, I. (2006). Issues in software engineering of relevance to instructional design. *TechTrends, 50*(5), 28-35.

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.

Dudziak, T. (1999). eXtreme Programming An Overview. *Methoden und Werkzeuge der Softwareproduktion WS, 2000*.

Eckstein, J. (2013). *Agile software development in the large: Diving into the deep*: Pearson Education.

Elssamadisy, A. (2008). *Agile adoption patterns: a roadmap to organizational success*: Addison-Wesley Professional.

Flick, U. (2015). Introducing research methodology: A beginner's guide to doing a research project: Sage.

Fontana, A., & Frey, J. H. (2005). The interview: From neutral stance to political involvement. *The Sage handbook of qualitative research, 3*, 695-728.

Fowler, M., & Foemmel, M. (2006). Continuous integration. *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*.

Fowler, M., & Highsmith, J. (2001). The agile manifesto. *Software Development, 9*(8), 28-35.

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.

Gable, G. G. (1994). Integrating case study and survey research methods: an example in information systems. European Journal of Information Systems, 3(2), 112-126.

Ghani, I., Izzaty, N., & Firdaus, A. (2013). Role-based Extreme Programming (XP) for secure software development. *Science International (Lahore), 25*(4 (Spe), 1071-1074

Gittins, R., Hope, S., & Williams, I. (2001). *Qualitative studies of xp in a medium sized business.* Paper presented at the Proceedings of the 2nd Conference on

eXtreme programming and flexible processes in software engineering, Cagliari, Italy.

Goldhor, H. (1972). Introduction to scientific research in librarianship: University of Illinois, Graduate School of Library Science.

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*.

Gulla, J. (2011). *Seven reasons why information technology projects fail.* Paper presented at the SHARE Conference.

Haider, M. T., & Ali, I. (2011). Evaluation of the Effects of Pair Programming on Performance and Social Practices in Distributed Software Development.

Haider, M. T., & Ali, I. (2011). Evaluation of the Effects of Pair Programming on Performance and Social Practices in Distributed Software Development.

Harrison, N. B. (2003). A study of extreme programming in a large company. *Avaya Labs*.

Hass, K. B. (2007). The blending of traditional and agile project management. *PM world today, 9*(5), 1-8.

Haughey, D. (2011). The Four Levels of Project Success–The Project Management Maturity Matrix. *URL: http://www. projectsmart. co. uk/four-levels-of-project-success. html, retrieval on*, *20*(11).

Hernon, P. (1991). The elusive nature of research in LIS. *Library and information science research: Perspectives and strategies for improvement*, 3-14.
Highsmith, J. (2000). Extreme programming.

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.

Highsmith, J. (2013). *Adaptive software development: a collaborative approach to managing complex systems*: Addison-Wesley.

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.

Hockey, L. (1984) The nature and purpose of research. In Cormack, D.F.S.(ed) *The Research Process in Nursing, (1st edn).* London: Blackwell Science, 1-10.

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.

Hussain, Z., Lechner, M., Milchrahm, H., Shahzad, S., Slany, W., Umgeher, M., & Wolkerstorfer, P. (2008). *Integrating Extreme Programming and User-Centered Design.* Paper presented at the PPIG'08: Proceedings of the 20th annual meeting of the Psychology of Programming Interest Group, Lancaster, UK

Ishak, I. S., & Alias, R. A. (2005). Designing a strategic information systems planning methodology for malaysian institutes of higher learning (isp-ipta).

Jeffries, R. (2003). Extreme Programming and Agile Software Development Methodologies: CRC Press LLC.

Jeffries, R., Anderson, A., & Hendrickson, C. (2001). *Extreme programming installed*: Addison-Wesley Professional.

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.

Kalermo, J., & Rissanen, J. (2002). Agile software development in theory and practice. *University of Jyväskylä*.

Kaplan, B., & Duchon, D. (1988). Combining qualitative and quantitative methods in information systems research: a case study. MIS quarterly, 571-586.

Kircher, M. (2001). *eXtreme programming in open-source and distributed environments.* Paper presented at the JAOO (Java And Object-Orientation) conference, Aarhus, Dinamarca.

Kircher, M., & Levine, D. L. (2000). The XP of TAO: extreme programming of large, open-source frameworks.

Kircher, M., Jain, P., Corsaro, A., & Levine, D. (2001). Distributed extreme programming. *Extreme Programming and Flexible Processes in Software Engineering, Italy*, 66-71.

Kongyai, B., & Edi, E. (2011). Adaptation of Agile Practices: A Systematic Review and Survey.

Koskela, J., & Abrahamsson, P. (2004). On-site customer in an XP project: empirical results from a case study *Software Process Improvement* (pp. 1-11): Springer.

Kruchten, P. (2013). Contextualizing agile software development. *Journal of Software: Evolution and Process, 25*(4), 351-361.

Kumar Srivastava, D., Singh Chauhan, D., & Singh, R. (2011). Square Model A Proposed Software Process Model for BPO based Software Applications. *International Journal of Computer Applications, 13*(7), 33-36.

Kuppuswami, S., Vivekanandan, K., Ramaswamy, P., & Rodrigues, P. (2003). The effects of individual XP practices on software development effort. *ACM SIGSOFT Software Engineering Notes, 28*(6), 6-6.

Lankshear, C., & Knobel, M. (2004). *A handbook for teacher research*: McGraw-Hill Education (UK).

Larman, C. (2004). *Agile and iterative development: a manager's guide*: Addison-Wesley Professional.

Layman, L., Williams, L., & Cunningham, L. (2004). *Exploring extreme programming in context: an industrial case study.* Paper presented at the Agile Development Conference, 2004.

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.

Lee, N. G. F. R. M. (1991). Using computers in qualitative research: Sage.

Leffingwell, D. (2010). *Agile software requirements: lean requirements practices for teams, programs, and the enterprise*: Addison-Wesley Professional.

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.

Lewins, A., Taylor, C., & Gibbs, G. (2005). What is qualitative data analysis (QDA). *Online QDA. Online: onlineqda. hud. ac. uk/Intro_QDA/what_ is_qda. php.{Accessed 19 July 2008}*.

Lincoln, Y. S., & Guba, E. G. (1985). *Naturalistic inquiry* (Vol. 75): Sage.

Lindstrom, L., & Jeffries, R. (2004). Extreme programming and agile software development methodologies. *Information systems management, 21*(3), 41-52.

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.

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

Macholz, C. W. (2007). *XP Project Management.* Master of Science, The University of Montana, United States.

Mannaro, K., Melis, M., & Marchesi, M. (2004). Empirical analysis on the satisfaction of it employees comparing xp practices with other software development methodologies *Extreme Programming and Agile Processes in Software Engineering* (pp. 166-174): Springer.

Marchesi, M. (2005). *Extreme programming and Agile processes in software engineering*: Springer.

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.

Marshall, C., & Rossman, G. B. (1999). Designing qualitative research Thousand Oakes: CA: Sage Publications.

Martin, R. C. (2003). *Agile software development: principles, patterns, and practices*: Prentice Hall PTR.

Maurer, F., & Martel, S. (2002). Extreme programming: Rapid development for Web-based applications. *IEEE Internet computing*(1), 86-90.

Maxwell, J. A. (2013). *Qualitative research design: An interactive approach* (Vol. 41): Sage.

McBurney, D., & White, T. Research methods/-Belmont (Calif.): Thomson/Wadsworth, 2007.–441 p: ISBN 0-495-09208-8.

McConnell, S. (2004). *Code complete*: Pearson Education.

McMillan, J. H., & Wergin, J. F. (1998). *Understanding and Evaluating Educational Research*: ERIC.

Miles, M. B., & Huberman, A. M. (1994). *Qualitative data analysis: An expanded sourcebook*: Sage.

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.

Mills, A. J. (2010). *Encyclopedia of case study research* (Vol. 1): Sage.

Mingers, J. (2001). Combining IS research methods: towards a pluralist methodology. Information systems research, 12(3), 240-259.

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).

Mohammed, H., & Rauf, A. (2015). Agile Project Management: Brief Review. *Lecture Notes on Software Engineering, 3*(3), 225.

Munassar, N. M. A., & Govardhan, A. (2010). A comparison between five models of software engineering. *IJCSI, 5*, 95-101.

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.

Nawrocki, J., Jasiński, M., Walter, B., & Wojciechowski, A. (2002). *Extreme programming modified: embrace requirements engineering practices.* Paper presented at the Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on.

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).

Omar, M., Abdullah, S., & Lailee, S. (2013). Agile practices: A cognitive learning perspective.

Omar, M., Syed-Abdullah, S.-L., & Yasin, A. (2010). Adopting Agile Approach: A Case in Malaysia.

Omar, M., Syed-Abdullah, S.-L., & Yasin, A. (2011). The impact of agile approach on software engineering teams. *American Journal of Economics and Business Administration, 3*(1), 12.

Päivärinta, T., & Smolander, K. (2015). Theorizing about software development practices. *Science of Computer Programming, 101*, 124-135.

Paulk, M. (2001). Extreme Programming from a CMM Perspective. *IEEE Software, 18*(6), 19-26.

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.

Pickard, A. (2012). Research methods in information: Facet publishing.

Pickering, C. (2001). Building an Effective E-project Team. *E-Project Management Advisory Service, Cutter Consortium, 2*(1).

Poppendieck, M., & Poppendieck, T. (2003). *Lean software development: an agile toolkit*: Addison-Wesley Professional.

Powell, R. R. (1997). Basic research methods for librarians: Greenwood Publishing Group.

Pressman, R. (2009). *Software Engineering: A Practitioner's Approach*. (7th ed.). New York, USA: McGraw-Hill Education.

Pressman, R. S. (2005). *Software engineering: a practitioner's approach*: Palgrave Macmillan.

Pressman, R. S., & David Brian, L. (2009). *Web engineering:: a practitioner's approach*.

Puvenesvary, M., Rahim, R. A., Naidu, R. S., Badzis, M., Nayan, N. F. M., & Aziz, N. H. A. (2008). *Qualitative Research: Data Collection & Data Analysis*: UUM press.

Qureshi, M. (2011). Empirical Evaluation of the Proposed eXSCRUM Model: Results of a Case Study. *International Journal of Computer Science Issues (IJCSI), 8*(3). 150-157.

Ragin, C. C. (1987). *The comparative method: Moving beyond qualitative and quantitative strategies*: Univ of California Pr.

Rejab, M. M., Omar, M., Mohd, M., & Ahmed, K. B. (2011). *Pair programming in inducing Knowledge sharing.* Paper presented at the Proceedings of the 3rd international conference on computing and informatics.

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.

Roulston, K. (2010). *Reflective interviewing: A guide to theory and practice*: Sage.

Rumpe, B., & Schröder, A. (2014). Quantitative survey on extreme programming projects. *arXiv preprint arXiv:1409.6599*.

Saldaña, J. (2012). *The coding manual for qualitative researchers*: Sage.

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.

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.

Satzinger, J. W., Jackson, R. B., & Burd, S. D. (2005). *Object-oriented Analysis and Design: With the Unified Process*: Thomson Course Technology.

Saunders, M., & Lewis, P. (2012). *Doing research in business and management: An essential guide to planning your project*: Financial Times Prentice Hall.

Schwaber, K., & Beedle, M. (2002). gilè Software Development with Scrum.

Sekaran, U., & Bougie, R. (2009). Research methods of business: A skill-building approach (ed.). New York: John Willey & Sons: Inc.

Sekaran, U., & Bougie, R. (2010). Research methods for business: A skill building approach. Wiley: London.

Senapathi, M., & Srinivasan, A. (2012). Understanding post-adoptive agile usage: An exploratory cross-case analysis. *Journal of Systems and Software, 85*(6), 1255-1268.

Sfetsos, P., & Stamelos, I. (2007). Improving Quality by Exploiting Human Dynamics in Agile Methods. *Agile Software Development Quality Assurance*, 154.

Sfetsos, P., Angelis, L., & Stamelos, I. (2006). Investigating the extreme programming system–An empirical study. *Empirical Software Engineering, 11*(2), 269-301.

Shore, J., & Warden, S. (2008). The Art of Agile Development O'Reilly Media Inc: Shroff Publishers and Distributors Pvt. Ltd.

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.

Sillitti, A., Succi, G., & Vlasenko, J. (2012). *Understanding the impact of pair programming on developers attention: a case study on a large industrial experimentation.* Paper presented at the Proceedings of the 34th International Conference on Software Engineering.

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.

Sinkovics, R. R., & Alfoldi, E. A. (2012). Progressive focusing and trustworthiness in qualitative research. *Management International Review, 52*(6), 817-845.

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.

Sison, R., Jarzabek, S., Hock, O. S., Rivepiboon, W., & Hai, N. N. (2006). *Software practices in five ASEAN countries: an exploratory study.* Paper presented at the Proceedings of the 28th international conference on Software engineering.

Skinner, M., & CIS, F. C. I. (2001). Enhancing an Open Source UML Editor by Context-Based Constraints for Components: University of Berlin, Thesis.

Sliger, M., & Broderick, S. (2008). *The software project manager's bridge to agility*: Addison-Wesley Professional.

Solinski, A., & Petersen, K. (2014). Prioritizing agile benefits and limitations in relation to practice usage. *Software Quality Journal*, 1-36.

Soundararajan, S., Arthur, J. D., & Balci, O. (2012). *A methodology for assessing agile software development methods.* Paper presented at the Agile Conference (AGILE), 2012.

Stamelos, I. G. (2007). *Agile software development quality assurance*: Igi Global.

Stellman, A., & Greene, J. (2014). *Learning Agile: Understanding Scrum, XP, Lean, and Kanban*: " O'Reilly Media, Inc.".

Stober, T., & Hansmann, U. (2010). *Best Practices for Large Software Development Projects*: Springer.

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.

Syed-Abdullah, S., Holcombe, M., & Gheorge, M. (2006). The impact of an agile methodology on the well being of development teams. *Empirical Software Engineering, 11*(1), 143-167.

Tan, S. (2011). How to increase your IT project success rate: Gartner.

Tessem, B. (2003). Experiences in learning xp practices: A qualitative study *Extreme Programming and Agile Processes in Software Engineering* (pp. 131-137): Springer.

Tian, Y. (2009). Adapting Extreme Programming For Global Software Development Project.

Tsvara, P. (2013). *The relationship between the management strategies of school principals and the job satisfaction levels of educators.* Doctoral dissertation, University Of South Africa, Pretoria.

Turk, D., France, R., & Rumpe, B. (2014). Assumptions underlying agile software development processes. *arXiv preprint arXiv:1409.6610*.

Turk, D., France, R., & Rumpe, B. (2014). Limitations of agile software processes. *arXiv preprint arXiv:1409.6600*.

Turk, D., France, R., Rumpe, B. (2002). Limitations of Agile Software Processes. *In Proceeding of the Third International Conference on extreme Programming and Agile Processes in Software Engineering held on 26-30 May 2002 at Alghero, Sardinia, Italy* (pp. 43-46). New York: ACM.

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.

Valacich, J., George, J., & Hoffer, J. (2009). *Essentials of system analysis and design*: Prentice Hall Press.

Vanhanen, J., & Korpi, H. (2007). *Experiences of using pair programming in an agile project.* Paper presented at the System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on.

Wells, D. (2009). Agile process. extreme programming: a gentle introduction.

Williams, L., Krebs, W., Layman, L., Antón, A., & Abrahamsson, P. (2004). Toward a framework for evaluating extreme programming. *Empirical Assessment in Software Eng.(EASE)*, 11-20.

Williams, L., Layman, L., & Krebs, W. (2004). Extreme programming evaluation framework for object-oriented languages. *Computer Science TR-2004-18*.

Willig, C., & Stainton-Rogers, W. (2007). *The SAGE handbook of qualitative research in psychology*: Sage.

Willig, C., & Stainton-Rogers, W. (2008). Qualitative research in psychology: Los Angeles & London: SAGE Publications.

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.

Wood, W., & Kleb, W. L. (2003). Exploring XP for scientific research. *Software, IEEE, 20*(3), 30-36.

Wu, B. H. (2011). *On software engineering and software methodologies a software developer's perspective.* Paper presented at the Information Science and Technology (ICIST), 2011 International Conference on.

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.

Xu, Y., Lin, Z., & Foster, W. (2003). Agile Methodology in CMM Framework: an Approach to Success for Software Companies in China. Proceedings of the GITM.

Yin, R. K. (2011). *Applications of case study research*: Sage.

Yousef Al-tarawneh, M., Syazwan Abdullah, M., & Bashah Mat Ali, A. (2012). Comparison of Extreme Programming (XP) method and key process areas of CMMI-DEV1. 2. *Global Journal on Technology, 1*.

Zikmund, W. (2003). *Business Research Methods the Dryden Press*: Harcourt College Publishers: Fort Worth.

Zuiderveld, N. R. (2003). *eXtreme Programming and SCRUM: A Comparative Analysis of Agile Methods.* Paper presented at the Published in the Proceedings of the International Conference on Software Engineering, Portland.