Extending Extreme Programming to Support Life Sciences Regulations

Hossein Mehrfard

A Thesis

In

The Department

Of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science at
Concordia University
Montreal, Quebec, Canada

April 2010

Your file Votre référence
ISBN: 978-0-494-67210-5
Our file Notre référence
ISBN: 978-0-494-67210-5

ABSTRACT

Extending Extreme Programming to Support Life Sciences Regulations

Hossein Mehrfard

The difficulty of complying with different regulations is becoming more evident as larger numbers of regulated businesses are mandated to follow a set of different regulations in their prescribed boundaries. This wide spectrum of regulations is inevitable as the result of today's heavily regulated society. In this thesis, we focus on the Food and Drug Administration (FDA) regulations and their impact on software development. FDA mandates stringent requirements on the process by which software systems for medical devices are developed. Many of these requirements involve documenting various activities in the software process, something that agile processes such as the Extreme Programming (XP) process do not support.

In this thesis, we discuss FDA requirements in detail and relate these requirements to software engineering practices. We then show areas where XP fails to meet these requirements. We propose an extension to XP that complies with FDA. The extended process is implemented using the Eclipse Process Framework (EPF). A questionnaire-based evaluation technique is used to validate the extended XP.

Acknowledgements

I would like to express my immense gratitude to Dr. Abdelwahab Hamou-Lhadj, my advisor, for his many contributions and unlimited support in this work. During the course of my Masters, his knowledge, insightful comment, and professional commitment inspired me to move forward in this project. His way of solving the problems taught me how to approach different problems deeply and methodologically, while his constant support and motivation encouraged me to complete this work.

My deepest thanks to my family for all they have shared and all the love they gave. All this would not have been possible without their continuous support.

Finally, I would like to express my sincere thanks to my colleagues for their valuable suggestions and timely efforts in my work with me.

Table of Contents

List of Tables

List of Figures

Chapter 1 - Introduction

1.1 Problem and Motivations

The recent advances in computing, networking, and medical device technology have resulted in a rapid growth of diagnostic and therapeutic devices. For example, the advanced imaging machines and micro devices for new surgical techniques (e.g. camera pills, computerized insulin pumps, implantable heart devices) are creating new opportunities for improved disease treatments. Despite this rapid growth of patient treatment systems, the lack of integration and interoperability among medical systems has often been the cause of insufficient health care delivery. This is conducive to an increase in health care cost and medical errors. To alleviate these issues, there is need to automate the control and management of medical devices through the development of safe and reliable software systems [1].

However, software systems used to control medical devices are subject to heavy regulations, known as life-science regulations, from government organizations to ensure that the design and testing of these systems have been carried out based on sound software engineering practices. Perhaps one of the most predominant set of regulations in North-America that regulates the way software systems used to control medical devices should be developed is the Food and Drug Administration (FDA) regulations. FDA is a U.S. government agency that regulates over $1 trillion of public health related products.

Many software development companies have to comply with FDA to deliver software solutions that operate in the medical and health sector.

FDA, however, imposes stringent requirements on the software process by which medical device software systems are developed. The FDA software process regulations have been developed on the basis of the premise that, with growing complexity of medical device software, it is difficult to assess the safety and reliability of software through traditional testing techniques. Additional verification and validation techniques as part of a broader and systematic process have to be applied. As such, FDA requirements often translate into documenting and following specific guidelines to certify that the system was built, verified, and validated in a systematic manner and according to proven software engineering practices. FDA requirements impact a large spectrum of software process activities including requirement analysis, design, implementation, maintenance, and testing phases [2].

One common approach to meet FDA regulations is to add a certification phase to an existing process in which a software compliance team gathers the necessary evidence to pass an FDA audit [3]. Although this approach has been shown to be useful for small projects, there are doubts about its applicability to larger and more complex systems, resulting in high risk of non-compliance. The problem is that gathering process information required by FDA might be costly and sometimes impossible if done after the fact (i.e. after the software has been produced). The information might simply not be available in the first place due to the fact that many existing software processes have not been designed with FDA requirements in mind. What is needed is to provide support for FDA requirements throughout the entire software process [1]. This requires extending

existing processes with additional practices and roles that fulfill FDA needs. This extension, however, is a challenging and complex task, especially for agile processes. There are two main factors that contribute to this complexity

- There are just too many FDA requirements to comply with. Many of these requirements tend be ambiguous and have been the source of inconsistencies [4].

- Agile software development processes tend to be light-weight processes that favour communication among team members in detriment of documentation. On the other hand, FDA requires written evidence that the process activities have been carried out properly. FDA also requires many additional process activities such as traceability analysis that are not supported by most agile processes.

The objective of this research is to tackle the above issues by studying FDA regulations for medical device software and assessing the capability of Extreme Programming (XP) to support these requirements. We chose to focus on XP, in this thesis, due to the fact that it is the first agile process that has been proposed and it appears to be the one that embeds the most values of the agile movement. We do believe, however, that the techniques presented in this thesis are readily applicable to other agile processes.

1.2 Thesis Contribution

The main contributions of this thesis are as follows:

- The study of FDA requirements for medical device software. These requirements cover a large spectrum of process activities including requirement analysis, design, implementation, and testing. We carefully study FDA regulations for

device software to extract these requirements in a way that they can be mapped to existing software process engineering concepts. We believe that this contribution can be used as a reference work for many organizations who struggle to meet FDA requirements due to their ambiguity.

- We study the capability for XP to meet FDA requirements for medical device software. We uncover areas of XP that need to be extended. We do this by proposing a mapping between XP and FDA for each software process activity.

- We propose an extension to XP that aims to meet FDA requirements while keeping the essence of XP values. For this purpose, we use the SPEM (Software Process Engineering Metamodel) to model the extended XP [5]. We also implement the extension in EPF (Eclipse Process Framework) to enable its adoption. A preliminary evaluation of the extended XP by two XP experts has also been performed.

1.3 Thesis Outline

The rest of the thesis is organized as follows:

- In Chapter 2, we present the background information needed to understand the content of this thesis. The chapter starts by introducing regulatory compliance and its impact on software development in general. Then, we introduce FDA regulations. The chapter continues with presenting XP, its strength and limitations. The chapter ends by presenting related studies that are close to the content of this thesis.

- In Chapter 3, we present an in-depth study of the FDA requirements for device software. The chapter starts by presenting the method we used to extract FDA process requirements. Then, for each process activity (e.g., requirement gathering, design, etc.) we discuss what is needed to meet FDA regulations. The chapter ends with a summary of these requirements.
- In Chapter 4, we map the FDA software process requirements to XP, our selected agile process. We clearly show areas where XP meets or does not meet FDA requirements. We summarize the mapping between XP and FDA at the end of the chapter.
- Chapter 5 focuses on our extension of XP to meet FDA requirements. The chapter also includes the evaluation of our extended XP. The chapter also covers the modeling of the extended XP using SPEM notation as well as its implementation using the Eclipse Process Framework (EPF).
- We conclude the thesis in Chapter 6 by revisiting the main research contributions. We also present future directions. Chapter 6 ends with our closing remarks.

Chapter 2 - Background

2.1 Introduction to Regulatory Compliance

Regulatory compliance is the act of complying with federal and state legal requirements, international standards, best practices, guidelines, and organizational policies and procedures [6]. Recently, there has been a significant increase of attention to regulatory compliance. This increase was driven by many factors including the latest corporate scandals which involve some of the major U.S. organizations (e.g., Enron, WorldCom), the reliance on Information Technology (IT) to protect and secure sensitive information, and a higher need for business continuity. According to an AMR research report [7], the cost of complying with different regulations had reached $28 billion in 2007. The consequences of not complying may include severe fines, lawsuits and possible imprisonment, customer dissatisfaction, and loss of market confidence [6]. For instance, in 2009, Eli Lilly and Company, the giant American drug company paid nearly $1.5 billion in fines for marketing one of its drugs that was not approved by FDA. The U.S. Department of Justice called this the "largest criminal fine for an individual corporation ever imposed in the United States" [8]. A similar case was reported in 2009, where Pfizer, the world's largest drug company, was fined to pay $2.3 billion as a result of promoting four drugs for use on people's ailments which were not approved by FDA [9]. Examples of most popular North-American regulations include FDA regulations (the

focus of this thesis), the Sarbanes-Oxley Act (SOX), the Health Insurance Portability and Accountability Act (HIPAA), and so on.

Regulatory compliance has a direct impact on the way software systems used by regulated organizations are developed. They do not only need to satisfy typical user functional requirements but they also need to be developed taking into account the applicable regulatory compliance requirements. For example, a software system that is used to manage patient records in a North-American health institution must satisfy the Health Insurance Portability and Accountability Act provisions which require the presence of several authentication and security mechanisms to protect patient information saved in an electronic format. Based on a survey conducted by Information Systems Audit and Control Association (ISACA) in May 2008, regulatory compliance is projected to be the key concern for the Information Technology (IT) industry for the next coming years [10]. The problem is that there are just too many legal requirements to comply with which vary from one industrial sector to another. This is made more complex for global software companies that operate at the global level and which are required to comply with international laws depending on the geographical area of their client base.

Although regulatory compliance can impact software development in many ways, in this thesis, we focus on compliance requirements, in particular compliance with FDA regulations, which impact the software development process by which software systems used in medical devices are developed.

2.2 FDA Regulations

The Food and Drug Administration (FDA) is a U.S. government agency that protects consumers by enforcing the U.S. Federal Food, Drug, and Cosmetic Act [11]. The FDA protects consumers by regulating and monitoring areas related to public safety and medication such as foods, tobacco products, medication drugs, medical devices, electromagnetic radiation emitting devices, etc. The FDA regulates more than $1 trillion worth of consumer goods, about 25% of consumer expenditures in the U.S. The FDA tends to issue severe fines for non-compliance, which makes its regulations some of the most important regulations that organizations need to consider in their compliance strategy.

The FDA also regulates the design and use of medical devices. There are several guidelines that have been issued by the FDA on how to monitor the manufacturing of safe and reliable medical devices. The software systems that control medical devices are also regulated under the FDA regulations. Due to complexity and criticality of medical devices, the FDA sets high demands on how to develop software for medical devices. Most of the FDA requirements for medical device software are directly related to the process activities used by the organization to develop the software. In addition, the FDA expects sufficient level of auditability within the software process. In other words, certain aspects of the development life-cycle need to be tracked to allow external auditors to assess whether the system is FDA-compliant or not.

2.3 Extreme Programming

A software process is defined as a set of activities, methods, practices, and transformations that are used to develop and maintain software and its associated products [12]. Since the 90's, a large number of software development processes have been introduced but only a few of them have gained wide utilization at the time [13]. In spite of the emergence of new software development approaches, building software is still a difficult task and the number of software project failures is still relatively high [14]. Agile software development principles and methodologies have gained a lot of popularity in recent years. Unlike traditional software processes, agile processes advocate flexible practices that allow coping with changing requirements. They have been developed by practitioners based on their experience working on several software development projects [15, 16]. Although existing agile processes can differ significantly in the way they approach software development, they all share one key characteristic which consists of favouring close collaboration between software development and business teams via face-to-face communication, as opposed to putting an emphasis on written documentation [12, 15]. In addition, agile processes propose a set of practices to deal with unstable and changing requirements. Although there is no clear definition of agility and which approach could be considered agile, the agile software development manifesto [17], a statement presented by a number of practitioners in 2001, states that an agile process should favour [13]:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation

- Responding to change over following a plan

Some of the advantages of agile processes include the fact that they are able to manage changing requirements during the development processes by assuming that the requirements are not deterministic and may change as the project evolves. To manage changing requirements, agile processes attempt to split and prioritize the requirements and implement them in a number of small iterations and releases.

There exist several agile processes including Extreme Programming (XP) [18], Scrum[19], Feature-Driven Development (FDD) [20], Crystal Methodologies (CM) [21], and so on. The one we want to focus on in this thesis is XP. To many people, XP has come to embody the agile methodology itself. This is, perhaps, because it is one of the first agile processes that has been proposed. In general, XP consists of a set of individual practices that when put together yield a successful software practice [13].

As illustrated in Figure 2-1, XP starts with exploration phase. Customer writes stories about the features of the system for the first release in collaboration with a programmer. A programmer leads the customer to think about the expected functionality of the system by asking specific questions. The functions of the system are then written in story cards. The story contains a list of tasks that need to be supported for better estimation and management of the requirements in later phases of the process. In addition, the development team studies the feasibility of the system and proposes possible architectures through building prototypes. The decision about the type of technology, tools and configuration management systems to be used is made during the exploration phase. This phase takes usually few weeks to few months [13, 22].

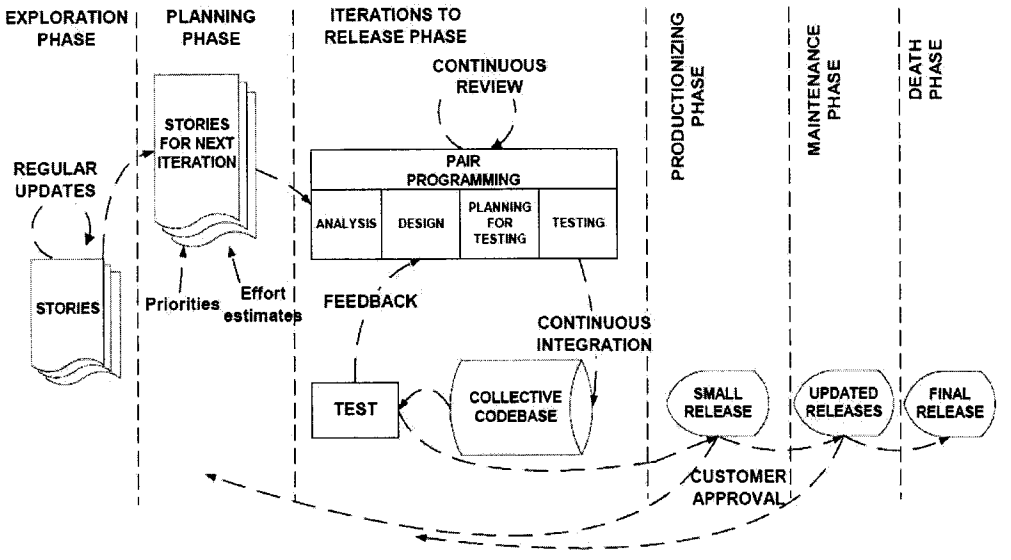

Figure 2-1. Extreme programming process life cycle (taken from [13])

The planning phase starts by prioritizing stories for the next small release based on their business values. The required time and efforts for the release is estimated. The first release takes not more than two months [13].

At the beginning of developing the release, customers select stories based on their business values and then the programmers break down those prioritized stories to a number of tasks and estimate for each the required time and efforts for each task. Based on the estimated tasks and the structural stories, the story cards are reprioritized again to produce the iteration plan for all the iterations [22]. It is important for the first iteration to be based on those stories that can reveal the system structure (architecture) for the consequent iterations [13]. During each iteration, the selected stories are implemented by a pair of programmers and tested using acceptance tests (functional tests). The iteration is not considered successfully implemented until it passes the acceptance tests, which are

normally written by the customer so as to verify that the system functionality satisfies the customer's needs.

During the productionizing phase, additional tests to assess the performance and quality of the release are applied. System tests are designed by a XP programmer (who occupied the role of architect) to examine that the architecture can be applied in this phase. In case there is a need for any additional changes to the current release, the release starts again from the planning phase with a maximum of one week duration for each iteration. Then the approved release is documented and deployed to customers [13, 22].

During the maintenance phase (i.e. after the release has been released), enhancement and correction to the deployed release take place for customer while the team is developing new release simultaneously [13].

When the customer is fully satisfied with the performance and the quality of the system and that all expected system features are implemented, the death phase is reached. No more changes are needed for system and all documentations are written. Another possible case for this phase happens when the project fails to illustrate the expected functionality or the cost of the project becomes too high to continue the project [13].

XP supports the idea that the design should be as simple as possible. To achieve this objective, XP puts an emphasis on using refactoring techniques such as removing duplicated code, improving the existing design. It should be mentioned that programmers must verify that the system is still operational after a refactoring activity takes place. The XP process requires that design, implementation, and testing of the system should be carried out by a pair of programmers sharing one computer. This allows programmers to spend more time finding solutions to a challenging problem and less time doing routine

debugging. Pair programming has been shown to be a useful technique for building robust software [23].

XP is a test-driven development method such that, before writing code for a particular story, programmers must implement the automated tests to verify the story functionality. The programmers rely on unit testing to verify the correctness of the story. The work on a story is not considered complete until it has been shown to be defect free. Integration tests are run to verify that the overall functionality of the system is bug free after an iteration [13].

Using XP, the resulting implementation is owned by all team members. This collective ownership of the artefacts of the system allows programmers to make modifications to parts of the code that have been created by others. The main advantage of this practice is to speed up the development process such that when programmers detect a fault in the code he or she has the right to fix it. A coding standard is used to make sure that the development team uses the same design and coding conventions. To keep the development team motivated, XP discourages team members from working more than 40 hours a week. In addition, overtime weeks are usually limited to no more than two weeks in a row [13].

In Table 2-1, we classify the XP process based on the roles, practices, and artefacts of XP. For this aim, we used the Eclipse Process Framework (EPF) and its process modeling library for XP [24].

Table 2-1. XP roles, practices, and work products

Role	Practice	Artefact
XP Coach	Adapt & improve process	
	Explain process	
	Improve team skills	
	Keep process on track	
	Resolve conflicts	
XP Customer	Adjust iteration scope	User story
	Define customer test	XP customer test
	Define iteration	XP iteration plan
	Define release	XP release plan
	Report customer test result	XP vision
	Report project status	
	Revise release plan	
	Write user story	
XP programmer	Break down story	Coding standard
	Define coding standard	Metaphor
	Estimate task	Production code
	Estimate user story	XP build
	Implement spike	XP unit test
	Integrate & build	System test for
	Refractor code	architecture
	Help customer for story writing	
	Usage Evaluation	
	Large scale refactoring	
	System partitioning	
	Write system test for architecture	
	Write code	
XP programmer(administrator	Setup programmer environment	
XP tester	Automate customer test	
	Run customer test	
	Setup tester environment	
XP tracker	Track iteration progress	
	Track release progress	

XP Coach:

Coach has as a managerial role in XP and is the person (or number of persons) who is responsible for whole process including guiding and leading other team members throughout the process [13, 22]. He is the main person who supports the customer team, development team, and the organization. The existing model for XP lists the expected practices (Figure 2-2) of the XP coach as follows: adapt and improve process, explain process, improve team skills, keep process on track, and resolve conflicts [24].

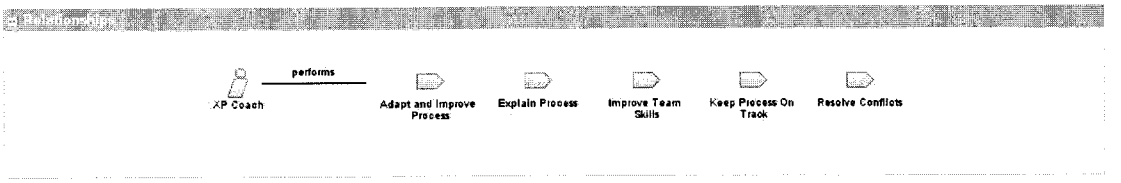

Figure 2-2. XP Coach (taken from [24])

XP Customer:

The XP customer or user has three main responsibilities (Figure 2-3) [24, 25]:

- Defining the characteristics of the product he or she expects from the development team to build by defining user story cards
- Indicating the sequence of building software by choosing stories for iterations and releases in planning phase as well as in iterations to release phase.
- Writing acceptance test for each story before the iteration starts.

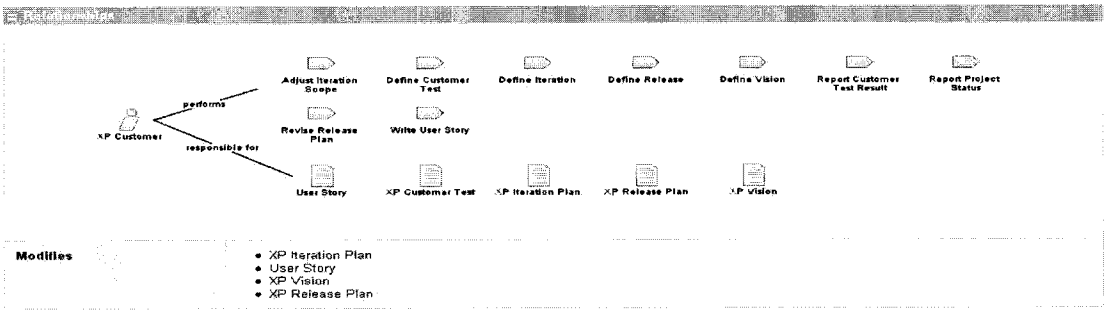

Figure 2-3. XP Customer (taken from [24])

XP Programmer:

An XP programmer is divided into many sub-roles that assume different, but related, roles. The XP architect sub-role is responsible for the architecture of system as the system grows. He or she is also responsible for large scale refactoring and writing system tests. The XP architect sub-role is a key role that directs other programmers to follow the architecture of the system and to define the system partitions that are the focus of various

iterations [18]. The XP interaction designer sub-role is responsible of to break down the story cards and estimate the required time and resources for each story card. He or she is also responsible for writing the metaphor of the system. He or she has to follow simple design as the XP value during writing the metaphors. In addition, the XP interaction designer helps customers write stories and analyze the usage of the actual system for possible new stories. Furthermore, he or she has to evaluate the usage of the deployed system. This evaluation may requires from the XP interaction designer to redefine the system interface during the project [18]. There is an XP implementer sub-role who is responsible of defining coding standards, implementing the tasks, and refactoring the code. The coding standard is a process artefact used by other roles. The XP programmer as the sub-role of XP programmer is responsible for writing the code and the unit tests. The XP integrator has to integrate every parts of the code coming from XP programmers and to develop XP builds [24]. The practices and work products of the XP programmer role (which embodies the above mentioned sub-roles) are illustrated in Figure 2-4.

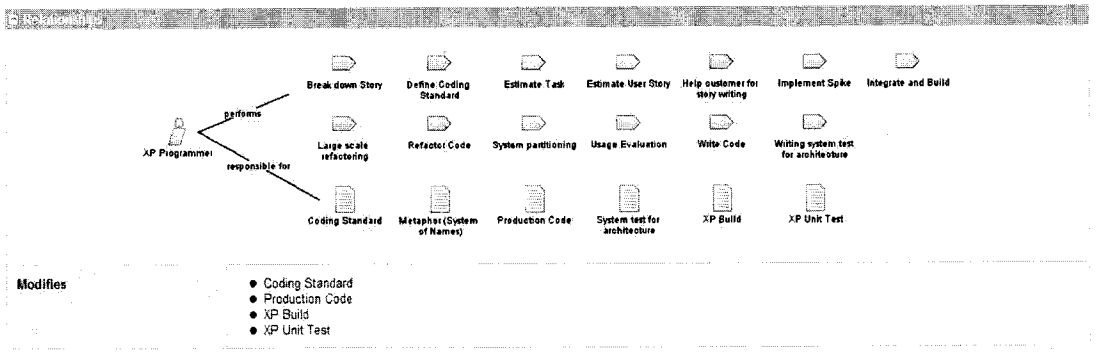

Figure 2-4. XP Programmer (adapted from [24])

XP Tester:

The XP tester is responsible for helping customers write and implement acceptance (functional) tests for each story card. He or she has also to help customers make decisions about the expected level of quality by defining acceptance criteria that can later be tested using acceptance tests. The tester is also responsible for automating customer acceptance tests, running the tests per each iteration, broadcasting the results, and preparing and maintaining testing tools [24, 26]. The role of the XP tester is shown in Figure 2-5.

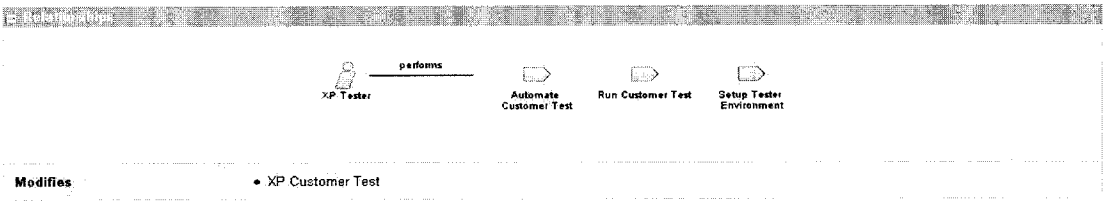

Figure 2-5. XP Tester (taken from [24])

XP Tracker:

The XP tracker is a managerial role that aims to keep the process in the right direction by gathering project metrics through regular feedback from the development team. He or she has to track the release plan (user stories), the iteration plan (tasks) based on the estimates made by the development team and evaluate the feasibility of these estimates for future improvements. Moreover, the XP tracker is responsible for tracking acceptance testing [13, 18, 24]. The XP tracker role is shown in the Figure 2-6.

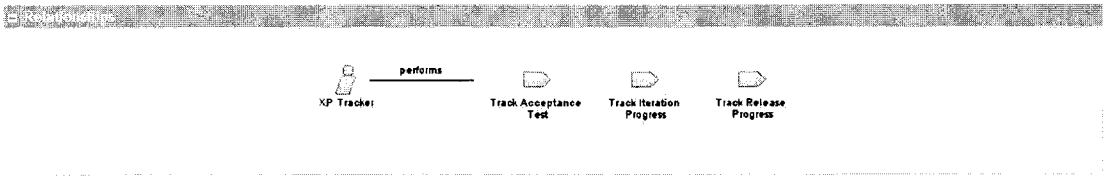

Figure 2-6. XP Tracker (adapted from [24])

2.4 Related Work

Kent Beck, one of the leading voices of XP, discussed the use of XP for developing secure and safe software in his recent book [18]. He points out that features such as safety and security have become the first priority requirements in developing software in areas like avionics and medical systems. He suggests believes that XP has sufficient capability to support developing such software systems only if additional practices for security or safety are incorporated into XP. He also argues that XP can be adapted to developing software for FDA medical devices by putting the emphasis on the audit process during the XP life cycle. He considers auditing as a continuous practice that starts early in the XP life cycle instead of having it as a separate phase at the end of project. However, it is not explained how XP can be extended to consider the FDA audit process, which is the topic of this thesis.

In [27, 28], McCaffery et al. have addressed the issue of the compliance of medical devices with FDA regulations at the process improvement level. For this, they suggested the application of a software process improvement process like CMMI to ensure FDA regulatory compliance. Our work differs significantly from theirs in several respects by focusing on how XP does not meet FDA requirements. We also proposed an extension to XP.

In [4], Abdeen et al. proposed to separate the notions of process and product in FDA guidelines. They believed that the lack of separation between software process and the results of the software process (i.e. product) caused ambiguity and inconsistency in FDA guidelines. To support their claim, they compared FDA guidelines with the ones of an

important security standard called "Common Criteria for Information Technology Security Evaluation". They believed that the methodology used in the Common Criteria resulted in clear expectations for both external auditors and software developers by providing objective criteria for both the process and the product.

In [29], Wright explained how he achieved ISO 9001 certification [30] using XP in his company. ISO 9001 requires having a quality management framework where business processes of the organization are documented and monitored. Wright proposed a light-weight extension to XP that meets ISO 9001 requirements. He first mapped ISO 9001 process requirements to XP practices. Then, he proposed a way to monitor and measure the process activities. For instance, he created virtual white board to add more features to XP stories and record them. In addition, he related integration, system, and acceptance tests to their corresponding virtual stories. The difference between Wright's work and the content of this thesis is that FDA requirements vary significantly from those of ISO 9001. The FDA is concerned with every single activity of a process.

Chapter 3 - FDA Process Requirements for Medical Devices

In this chapter, we start by presenting our approach for extending a software process to support FDA regulations. Then, we show the application of this approach for extracting the software process requirements for medical devices from the regulations and guidelines provided by the FDA. This requirement extraction is done based on traditional software development phases.

3.1 An Approach for Extending Software Methodologies to Support FDA Regulations

Many Life-Science Regulations (LSRs) such as FDA regulations establish guidelines for software development in variety of products. External auditors seek for evidence that shows that the development team has complied with those guidelines during the development process.

The FDA often defines the guidelines in a holistic way that is generic enough to be applied to various development methodologies. Unfortunately, this can cause ambiguities for software developers since no specific development methodology can abide by the provided guidelines. For example, the FDA requests the medical device software developers to build safe and reliable software while no specific quality criteria are explicitly provided by the FDA.

Furthermore, the FDA often uses terms that are not specific to software engineering. That is, a single term can be used in more than one field while having many completely different meanings. For instance, "risk analysis" can refer to an activity in both software requirements engineering and project management. This also can cause confusion in the intended meaning of a term making it difficult for development companies to comply with these guidelines as the developers do not know what they specifically have to follow.

In our approach, we address this problem by following a framework through which we can extend a software development methodology of interest in a way that it can support a life science regulation. This framework (Figure 3-1) is composed of the following steps:

1. We go through all the LSR guidelines and extract the guidelines related to software development.

2. We study these guidelines from the software engineering process perspective and present typical software practices and documentations that can help developers follow the guidelines.

3. According to the suggested practices and documentation, we investigate the capabilities of our desired software development methodology for supporting these requirements.

4. Based on our evaluation on how well our desired software development methodology can support the extracted requirements, we propose an extension of the methodology to support the missing requirements.

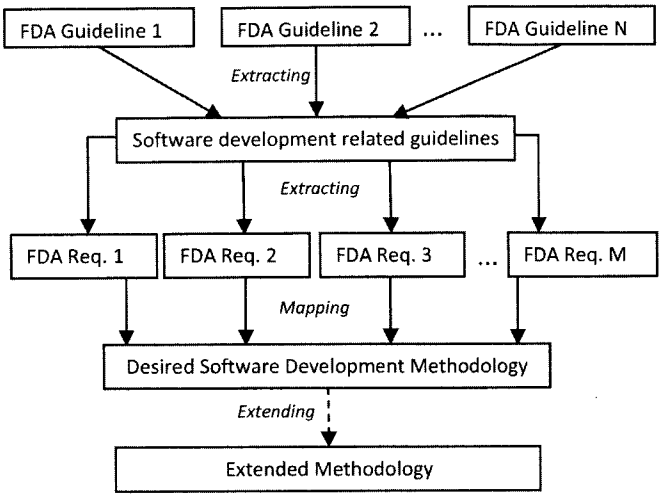

Figure 3-1. The framework on how to extend a software development process to meet FDA

3.2 Software Process Activities

The FDA regulations, is an LSR that imposes stringent requirements on the process by which software systems used in medical devices are developed [2, 31-33]. These requirements translate into various software artefacts that must be made available for the software to be FDA-compliant. In this Section, we discuss these requirements in detail. First, we went through the FDA guidelines associated with medical devices and extracted the guidelines related to software development activities. Then, we studied these guidelines from the software engineering processes perspective and presented typical software practices and documentation that can help developers follow FDA guidelines.

We classified FDA software development requirements into four phases: Requirement, Design, Coding and Construction, and Testing. For each phase the required FDA practices are detailed.

3.2.1 Requirements Gathering and Analysis

The FDA pursues number of goals in requirement phase of medical devices. This phase has to be contained steps for identification, analysis and documentation of information about the medical device and its intended use. Towards those goals, some areas gain heavier importance such as: "allocation of system functions to hardware/ software, operating conditions, user characteristics, and potential risks". This has to result in clear statement of the use of intended software [2].

3.2.1.1. Requirements Elicitation

The FDA requires at this stage a complete documentation that clarifies all the software inputs, Software Requirements Specification (SRS) and all the expected software outputs. For instance, FDA wants developers to illustrate all ranges, limits and default values which are acceptable as the software inputs, the clear functionality of software in computer-based system and the expected results of applying those inputs throughout software functionality as software outputs [2].

Discovering requirement specification and identifying software inputs and outputs make this stage similar to requirements elicitation in software requirements engineering. Documenting software inputs and outputs helps to understand the boundary of the system which is necessary for requirements elicitation [34].

The FDA also demands that non-functional requirements (NFR) such as performance, reliability, security and the safety features of software be defined clearly. A particular

emphasis is put on safety requirements as unsafe medical device may cause loss of human lives [2, 32].

In addition, the FDA requires that all communication points between the software in question and other software systems, hardware, and persons be well defined. These communication points are known as the interfaces. For instance, the communication point between the software and users is the user interface [2, 35].

The Human Factors Engineering (HFE) is one of the key requirements mandated by the FDA during the software development process. The HFE activities are more centered around software design. For this reason, the HFE process is explained in more detail in the software design phase.

The FDA requires recognizing HFE requirements in software requirements phase. FDA suggests many activities including observations, interviews, and conducting focus groups to understand HFE requirements [33]. In addition, the characteristics of users of the system should be defined in the software requirements phase.

Software Requirement Specification (SRS) should describe all the user characteristics based on user's knowledge, ability, expectation, and limitation. Therefore, developers should investigate the experience of the users with similar software as well as the users' sensory capabilities (vision, touch, and hearing). User characteristics are highlighted by the FDA due to the nature of medical devices and its impact on software system design such as user interface design, safety and reliability requirements [2, 35].

In software engineering, some sorts of activities are applied to fulfill requirements elicitation such as interview, use case, observation and social analysis, focus group, brainstorming, joint application development(JAD), requirement modeling and prototyping [34, 36].

3.2.1.2. Requirements Evaluation

The FDA seeks for requirements evaluation policy and documentation in Development Company. The purpose of requirement evaluation is to resolve the incomplete, ambiguous, inconsistent and conflicting requirements. In addition, the requirements should be evaluated against possible risks. The FDA considers the possibility of applying requirements evaluation in multiple steps (incrementally) during development. The FDA expects developers to achieve complete, clear and non risky functional and non functional requirements by having a mechanism for requirements evaluation [2].

Requirement risk analysis is part of risk management document which focuses on potential risks of requirements that may cause failure in software. The FDA also emphasizes the analysis of risks coming from HFE requirements in software. Risk management is planned and conducted before entering the requirement phase on project level [2]. FDA suggests the formal review of requirements before starting extensive software design [2].

From software engineering perspective, the term requirement evaluation is similar to requirement analysis and validation in software requirements engineering. The numbers of activities are using to evaluate requirements. These activities are: formal review meetings, risk analysis, requirements inconsistency management, requirements

prioritization, evaluation of alternative options in requirements, requirement verification, and prototyping [34, 37].

To analyze the requirement risks, risk management techniques should be applied to requirements. Different approaches are proposed to handle risks in process and project scale of software engineering. Van Lamsweerde suggests the discipline for requirements' risk management which is divided to identify, assess and control the risk. For risk identification, risk checklists, component inspection, risk trees, and elicitation techniques like scenarios are considered. For risk assessment, qualitative and quantitative assessment techniques are applied. Finally for risk control, the countermeasure should be explored and the most appropriate countermeasures should be selected [38].

3.2.1.3. Requirements Traceability Analysis

The FDA believes traceability analysis is the essential activity during the entire development process. Traceability analysis defines the relationship between the software development products to keep the logical order of those products. This logical order among software artefacts becomes more evident when we are passing from one development phase to another phase [2, 35]. The general traceability relations during the development process is shown in Figure 3-2.

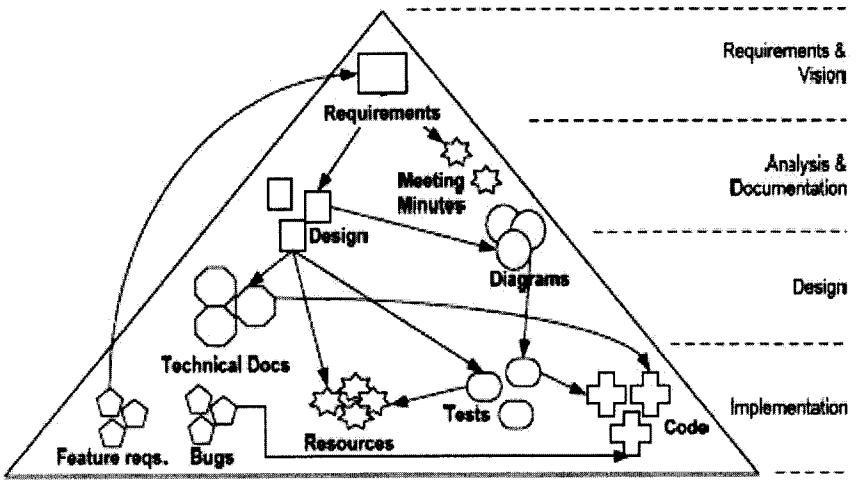

Figure 3-2. Traceability procedure (taken from [39])

The FDA puts an emphasis on traceability analysis during the requirement phase by establishing the relationships between following parts:

- Software requirements and system requirements (and vice versa)
- Software requirements and the risk analysis results

All those requirements of computer-based system (e.g., hardware platforms, operating systems, controller software and other application software) which are in relation with the requirements of software, must be defined clearly. In addition, the relationship of the requirements with recognized risks coming from the risk analysis results must be determined. As mentioned before, risk management is conducted at the project level before the requirement phase starts. The result of the project risks and the risks which are uncovered during the requirement phase should trace back to requirements [2, 35].

In software requirement engineering, requirement traceability analysis is considered as one of the activities of requirement management. Requirement management is usually

supported by CASE tools during software development. To achieve this, different traceability matrices are suggested such as source traceability, requirement traceability, design traceability and other traceability matrices based on the expected level of quality [37]. A traceability matrix is a table that shows bidirectional relations between the artefacts of two different phases such as a requirement-to-design matrix or a test cases-to-risk analysis traceability matrix.

Hyperlinked documents are another way for representing traceability besides traceability matrices. In a hyperlinked document, traceability statements are highlighted and linked to other statements in order to traverse them in both directions [40].

While FDA expects traceability matrices to show the relations between system requirements and software requirements as well as system requirements and the result of risk analysis, there is no practice in XP to support the creation of traceability matrices.

3.2.1.4. Test Plan

At this stage, FDA requires to develop the system and acceptance test plans. According to ANSI/IEEE standard 829, a test plan is defined as "the documentation of scope, approaches, resources, and schedule of intended testing activities. This document should identify test items, the features to be tested, the testing tasks, responsibilities, and any risks requiring contingency planning" [2, 35].

An acceptance test plan is documented based on a set of acceptance criteria provided by the customer to approve the final product. It is usually created through a close collaboration between customers and developers [2, 41].

A system test plan is written with respect to criteria for testing the software product on a specific operating platform to detect performance issues, and situations of stress [2, 41]. XP requires from customers to explain the acceptance criteria of software before each release [22]. These acceptance tests could be executed by a customer or a developer. FDA requires an acceptance test plan before starting implementing story cards. In XP, the acceptance test plan which is known as the customer test document is defined by the customer [22]. Then these tests are automated by testers to generate acceptance tests.

System testing is supported in XP at the end of each release. The XP programmer (who plays the role of an architect) is responsible for performing system testing [18]. A System test plan is written by an architect and put in practice with the help of an XP tester.

3.2.1.5. Configuration Management and Software Requirements Specification

The FDA requires a configuration management system to control and archive the produced software requirement specification documents. The objective is to keep the history of SRSs to ensures that the change are handled systematically [2].

Despite the existence of collective code ownership and continuous integration in XP there is no guideline in XP to support version control [42].

The FDA practices and documentations in the requirements phase are summarized in Table 3-1.

Table 3-1. FDA medical devices' requirements in software requirements phase

FDA Requirements - Requirement Phase -	Typical Practices	Typical Documentations
Requirements Elicitation	Interviews, use case analysis, observation and social analysis, focus group, brainstorming, requirement modeling, prototyping	Software Requirements Specification (SRS)
Requirements Evaluation	Formal review meetings, risk analysis, requirements inconsistency management, requirements prioritization, evaluation of alternative options in requirements, requirement verification, prototyping, requirements risk analysis	Result of the evaluation needs to be documented
Requirements Traceability Analysis	Create traceability matrices	Software requirements and system requirements traceability matrix, software requirements and the risk analysis result traceability matrix
Test Plan	Working on acceptance and system test plans	Acceptance test plan, system test plan
Configuration management	Version control	Not Specified

3.2.2 Design

The FDA defines the design phase as the process of translating the user requirements into their related logical components to be implemented. Due to complexity of medical devices, it suggests both high-level and detailed design. The FDA requires software design specification (SDS) document in the design phase. This document should contain

coding guidelines to help programmers during the coding phase. In addition, it should cover the following software development activities [2].

The software design in XP is done during the "iterations to release" phase by pair programming. The design in XP should be kept as simple as possible. The architecture as the high-level design is designed during exploration and planning phases and built during the first release in the "iterations to release" phase. Then detailed design is provided based on selected story cards, which is later implemented [13, 18]. In addition, there is no documentation in XP that represents a SDS. The design sketches are recoded informally by methods such as taking photo of design on whiteboard [22]. There is the need in XP to follow coding standards during the "iterations to release" phase. The coding standard is defined by chief programmer in a separate document.

3.2.2.1 Design for Human Factors Engineering

The FDA highlights the importance of Human Factors Engineering (HFE) in the design process by defining HFE as a discipline that should be taken during software and hardware design to improve human performance in using medical equipments based on their abilities [33]. The FDA recognizes that the design for safety of medical devices should take into account human factors. The reason is that according to the FDA Center for Devices and Radiological Health (CDRH), the lack of attention to human factors during product development may lead to errors that can potentially cause serious patient injuries or even death [33].

In [31, 33], FDA specifically describes human factors engineering and the policy required to take in medical device design in addition to [2] that emphasizes HFE during software

design. The FDA defines three areas to capture human factors in medical projects which are medical device, user, and use environment [33].

The FDA medical device guidelines propose a set of requirements for HFE in medical device projects. In our study, we considered medical device projects as computer based system projects due to their software and hardware requirements.

The FDA provides a number of guidelines on how to deal with HFE on software design. To reach the HFE in software design, the FDA suggests "following Human Computer Interface (HCI) guidelines", "improving software usability", and "performing software design coordinated with hardware design". Here, we only consider the "improving software usability during software design" requirement as an important requirement. To improve software usability, the FDA suggests a number of usability tests such as scenario-based testing, and testing the product by users per iteration of software development [33].

In software engineering, software usability engineering is a discipline defined to improve the usability of software, which is defined on seven characteristics which are understandability, learnability, memorability, efficiency, low error rate, compliance to standards and guidelines, and user satisfaction [43, 44]. These characteristics should be evaluated through inspection and usability testing. The methods for usability inspection include heuristic evaluation, cognitive walkthroughs, pluralistic walkthroughs, perspective-based inspection, and standards inspection/guideline checklists [44].

As the focus of FDA is on evaluating usability during the design phase, none of the above evaluation techniques in usability testing and inspection supports usability assessment

during software design. Rather, they evaluate usability based on a produced user interface or a prototype [44].

Recently, there has been an increase in the number of usability analysis techniques that are capable to assess usability of software during the design process [44]. Moreover, there are set of design solutions such as usability design patterns that can increase the usability of a software application [45]. The iterative nature of development in XP and the focus on user by getting constant feedback, and the high communication with customers makes XP project centered on user needs [46, 47].

During the exploration phase, XP does not suggest ways to deal with usability at the architectural level. To design the software system architecture, XP suggests building a system prototype during the exploration phase to evaluate possible architectures of software, create the high level design (architecture) of software during the exploration and planning phases, and finally the architecture is consolidated in the first release [13, 18]. There are two practices in XP that affect the design of the system architecture: system metaphor and simple design. System metaphors are shared stories to describe how the system works and simple design makes easier to understand each design component [48]. The XP is concerned with end users of software product by defining the role of the interaction designer. The Interaction designer as the sub-role of the XP programmer is responsible for evaluating the usage of the deployed system. This evaluation results are used to specify future functionalities of the system by defining additional user stories. In addition, the interaction designer refines the user interface according to the usage evaluation which is developed during several iterations to release phases [18].

3.2.2.2 Software Design Evaluation

The software design evaluation is considered as an integral part of the design process. The objective is to validate correctness, completeness, consistency, and maintainability of the design [2]. The FDA defines two categories of design evaluation activities: Design review, design verification and validation [32]. During the evaluation of design, activities such as analysis of control flow, data flow, complexity, timing, memory allocation, and criticality analysis should be supported [2]. Moreover, the FDA emphasizes on analysis of component interfaces in design evaluation to ensure all the defined interfaces in the requirement phase suit well the proposed design.

Design review meetings are required by FDA to support the fact that a design inspection has taken place. The focus of these meetings is to identify different concerns of software design and their potential side-effects, possible solutions, and the corresponding corrective actions in software design. During these meetings, designers present their design to the design reviewers. There are three types of design review: preliminary design review, critical design review, and system design review. The process is conducted in an iterative manner until potential problems are explored and solutions have been proposed [2, 32, 35].

The FDA defines design verification as a confirmation by examination that a specific requirement has been fulfilled. This requires documenting the design verification process. FDA suggests using some verification techniques such as fault tree analysis, and worst case analysis. In design control document, the FDA references the ISO 9001:1994 standard, where activities such as prototype evaluation, demonstration, simulation and

comparing the design with other similar proven designs are considered as software design verification activities [30, 32].

Design Validation is the examination that a design responds to user needs or the intended use of the product. The FDA requires documenting the design validation process. Design validation should be executed under actual or simulated use condition. It also needs to follow a successful verification to ensure that each user requirements is fulfilled. For this purpose, the FDA requires to provide a validation plan, validation methods and validation review. Some of the design validation techniques recommended by FDA include analysis and inspection methods, compilation of relevant scientific literature, and provision of historical evidence that similar designs are clinically safe [32].

3.2.2.3. Design Traceability Analysis

As mentioned in the requirement phase, the FDA requires traceability analysis throughout the entire development process. In the design phase, tractability analysis is conducted to verify that the entire design components are traceable to the software requirements and that all requirements can be mapped to a software design [2]. For this reason, there should be a design traceability matrix which relates software requirements documents to design specification.

3.2.2.4. Updating the Test Plans

The FDA requires to update existing test plans by generating module and integration test plans during the design phase. A module test plan should be created to test specific units

of the system. On the other hand, the integration test plan should be updated to test the flow of control and data between program units [2].

In addition, the acceptance and system test plans, which are created during the requirements phase, should also be updated considering HFE criteria defined in software requirements and design.

3.2.2.5. Test Design Generation

After preparing test plan, FDA asks development team to start generating test procedures and test cases for unit, integration, system and acceptance test based on the requirement and design of software that has done. These tests should be completed and finally executed in coding and test phases [2].

3.2.2.6. Configuration management and software design specification

Similar to the requirement phase, a version control system is needed during the design phase of the development process. What the FDA expects at this stage is the conformance of the software design specification (SDS) with established software configuration management. This emphasis comes from the fact that many development processes operate in an iterative way and produce different versions of SDS [2].

Table 3-2 summarizes the typical practices and documentations in design phase.

Table 3-2. FDA medical device requirements in software design phase

FDA Requirements - Design Phase -	Typical Practices	Typical Documentations
Design for Human Factors Engineering	Usability testing, usability inspection, usability inquiry, usability design patterns, scenario-based assessment of architecture	Documentation on design decisions that relate to making the system more usable
Software Design Evaluation	Prototype evaluation, demonstration, simulation, comparing the design with other similar proven designs, analysis and inspection methods, compilation of relevant scientific literature, provision of historical evidence that similar designs are clinically safe	Design review document, design verification document, design validation document, Software Design Specification (SDS)
Design Traceability Analysis	Create traceability matrices	Requirement-to-design traceability matrix
Update Test Plan	Working on unit and integration test plans	Unit and integration test plans
Test Design Generation	Generating test cases for unit, integration, acceptance and system	Test cases, test procedures
Configuration Management	Version control	Not Specified
Update risk analysis	Risk analysis of software design using techniques such as medical device use description to detect the device risks	Not Specified

3.2.3 Coding and Construction Activities

In this phase the detailed design specification should be implemented as a computer program. The construction is done either by directly start programming or assembling code components (e.g. using Commercial-Off-The-Shelf (COTS) products or code libraries). The selection of programming language and builder tools (i.e. assembler, linker or compiler) is considered very important by the FDA for development team. This decision should be taken regarding subsequent quality evaluation tasks such as availability of debugging and testing tools [2]. The FDA emphasizes on having following sub-phase during coding and construction.

3.2.3.1. Source Code Evaluation

Source code is evaluated before compilation to make sure whether the source code follows design specification and coding standards. The FDA underlines using desk checking techniques to evaluate the software code. Desk checking is a set of dynamic analysis techniques that can apply to evaluate the code [35]. These methods are code audit, code inspection, code walkthrough, and code review. In code inspection, the author of the code explains what the code supposes to do statement by statement in the meeting to analyze the program logic and its conformance to coding standards. In code walkthrough developers manually trace the source code with small set of test cases in the meeting to analyze the programmer's logic and assumptions. Code audit is a review of the source code by an independent person, or team to make sure that source code follows software design and programming standards. A code review consists of organizing

meetings where the software code is presented to project personnel, managers, or customers for feedback and approval [2, 35].

3.2.3.2. Source Code Documentation Evaluation

The FDA requires documenting the coding and the construction process. In most software projects, the commented code and the generated html or text files from the source code by tools are considered sufficient for documenting the code. However, the FDA requires documentation for each implemented module or function to show its agreement with coding standards and quality policies, defined within the organization. In addition, the existing errors after coding and construction have to be documented. Moreover, the whole process of compilation should be documented (happened errors, solutions and unsolved errors and warnings) [2].

3.2.3.3. Code Traceability Analysis

The FDA requires to have a traceability matrix to show the relation between the source code modules and the design specification and vice versa [2].

In addition, the traceability matrices to show the relation of the test cases and the source code as well as the test cases and the design specification is also needed [2].

3.2.3.4. Source code Interface Analysis

The implementation of the interfaces between the system modules should be clearly specified in the source code to ensure that the implemented communication links are well

integrated with the software implementation. This increases the safety of the final software product.

Table 3-3. FDA medical device requirements in software coding and construction phase

FDA Requirements - Coding Phase -	Typical Practices	Typical Documentations
Source Code Evaluation	Code audit, code inspection, code walkthrough, code review	Documentation that shows that code has been reviewed
Source Code Documentation Evaluation	Although there is no specific practices defined in the FDA guidelines, the FDA requires that the source code be documented and that the evaluation of this documentation should be performed	Source code document
Code Traceability Analysis	Create traceability matrices	traceability matrices for: source code to design specification, test cases to source code, test cases to design specification, test cases to risk analysis results, source code to risk analysis results
Source code Interface Analysis	Interface checking	Documents that show that interfaces between the system components have been verified
Test Generation	Updating test cases for unit, integration, acceptance and system testing	Document that describes test cases and test procedures

3.2.3.5. Test Generation

Besides the test procedures and test cases created to test the software design, the new test cases and their corresponding test procedures are generating based on the implementation. The new test cases can be unit, integration, acceptance, and system testing.

Table 3-3 summarizes FDA requirements for the coding and construction phase.

3.2.4 The Testing Phase

The FDA extensively focuses on software testing during the development process to ensure the reliability and safety of software product. It lists a number of software testing principles to examine software effectively such as the importance of what is to be tested rather than how to apply the test, having expected result for test, independence of test from coding, and the importance of test documents. The FDA highlights four types of testing activities: structural testing, functional testing, statistical testing, and regression testing [2, 35].

Structural or white box testing evaluates the internal code structure. The amount of structural test coverage is defined based on common metrics such as coverage of statements, branches, loops, conditions, and data flows [2].

Functional testing is a black box testing technique which is conducted to evaluate the program functionality and program interfaces. The FDA divides functional testing into four different types: normal case, output forcing, robustness, and combinations of inputs [2].

Regression testing is another type of testing technique to manage the changes during the software development life cycle. Regression testing ensures that changes to the system do not negatively impact the other parts of software [2, 35].

3.2.4.1. Test Documentation

Documenting test activities is an important concern for FDA during the testing process. The documents that FDA requires during the testing process include a test plan, test procedures, test cases, test reports, and test logs [2, 35].

Test plan, as defined in the requirement phase, should be created early in process to identify the testing tasks during each development stage. A test procedures document is generated from the test plan. It contains instructions about each test on how to setup the test and evaluate the test results. Test cases are designed and implemented depending on the type of testing (i.e. structural, functional, statistic, and regression). This document should identify system inputs, expected results, and a set of execution conditions for test. A test log is defined as a record of the test execution [35]. For instance, all detected errors during test execution should be logged. Once test execution is finished, test report should record the direction and results of the test. Test report should conform to requirements of its test plan. This report should help testers on their on their objective pass/fail decision [2, 35].

3.2.4.2. Test Execution

The FDA requires to perform the following testing activities: unit testing, integration testing, system testing and acceptance [2].

In unit testing, the program is divided to smaller components (modules). Then, the structure and functionality of each component is examined early in program testing [2].Integration testing is one level higher than unit testing. Integration testing concentrates on the flow of control and data between units [2].In system level testing, all aspects of functionality and performance of software product are tested. This test is done on working software products and developers should consider the requirements that exist at the level of the operating environment. The FDA highlights some aspects of software to examine within system testing such as performance issues, responses to stress conditions, security features, effectiveness of software recovery, HFE and usability, accuracy of documentation, and compatibility with other software products [2].

Finally, for user site testing, the FDA requires the conduct of user site testing as the last step of the testing activity of the software product. It defines user site testing as any testing through actual or simulated use of software as the part of installed system configuration at the user's site. As mentioned in the requirement phase, the FDA assumes that user site testing is the same as installation testing, beta testing, site validation, installation verification, and user acceptance test.

3.2.4.3. Test Traceability Analysis

The FDA requires several traceability matrices to link unit tests to detailed design, integration tests to high-level design, and system tests to software requirements [2].

There is no need for a traceability relation for acceptance testing (user site testing), since it is the customer who is responsible for this test and she or he is not aware of the details of the requirement and design. In addition, acceptance testing does not have a direct

relation with previous artefacts, because it tests the acceptance criteria such as performance and safety which are difficult to map to requirements, design, and code.

Table 3-4 summarizes FDA requirements in software testing phase.

Table 3-4. FDA medical devices' requirements is software testing phase

FDA Requirements - Software Testing -	**Typical Practices**	**Typical Documentations**
Test Documentation	FDA requires documenting the testing process	Test plan, test procedure, test case, test report, test log
Test Execution	Execution of unit tests, integration tests, system tests, and user site testing	Document that describes the results of executing the tests
Test Traceability Analysis	Create traceability matrices	Traceability matrices for: unit tests to detailed design, integration tests to high level design, and system tests to software requirements

Chapter 4 - Impact of FDA Life Science Regulation on Extreme Programming

In this chapter, we discuss the impact of FDA on XP by putting an emphasis on areas of XP that do not meet the FDA requirements. Similar to the previous chapter, this chapter deals with all phases of the software development life-cycle.

4.1 Mapping FDA Process Requirements to XP

4.1.1 Mapping to FDA Requirements

4.1.1.1. Mapping FDA requirements elicitation to XP

The XP provides a number of techniques to elicit requirements. The use of story cards is the technique to capture expected features of user. User stories are written by customers with the help of XP Programmer (interaction designer). User story is comparable to use cases since both aim at capturing the user's needs. However, a user story provides less detailed and more business valued customer features [34]. The FDA requirements for user characteristics of the system should be investigated during the writing of story cards. The developer should help the customer to define the major characteristics of the end users that are elaborated in more details during the next phases [49].

The high involvement of customers in XP is another effective factor for eliciting requirements. After writing user stories, the customer is involve during the "iterations to release" phase and helps break down the user story into multiple tasks. In addition, Brainstorming is another elicitation technique that is encouraged in XP with the help of customers and domain experts [34]. This customer involvement also results in better definition of HFE requirements.

As discussed in the previous chapter, the FDA requires defining and documenting the Software Requirements Specification (SRS), which should contain external interfaces, functional requirements and non functional requirements. However, there is no specific activity in XP that deals with different interfaces of a computer-based system. The functional requirements in XP are implemented by developers during the iterations to release phase based on user stories. The non-functional requirements, on the other hand, are dealt with in XP the same way as the functional requirements, i.e., during the iterations to release phase. There is common belief that XP tends to neglect the representation of non-functional requirements during the requirement gathering phase. This is due to the fact that XP tends to focus more on the functionality to provide to customers and less on non-functional requirements such as safety, portability, maintainability or performance. These requirements are only dealt with during implementation. However, usability is a common exception due to the constant customer feedback that the development team receives [34, 50, 51]. The requirement documentation level in XP is less than what is expected by FDA. XP, for example, does not require the presence of an SRS. The XP as an agile process tends to maximize team communication and minimize documentation by focusing on core deliverables of the

software system. The user story is the only document during requirement elicitation stage that could be used as the requirements' document when using the XP process.

4.1.1.2. Mapping FDA Requirements Evaluation to XP

Due to the incremental nature of XP, requirements are evaluated within different iterations and releases. In fact, the product resulting from an XP iteration or XP release is seen in XP as a prototype that can be evaluated by customers [13].

Requirement prioritization is a constant practice in XP during the planning phase and the iteration to release phase. During the planning phase, the customer selects the story cards for the next release based on its business values which is documented in the release plan. He is also responsible for choosing the story cards for each iteration during the iteration to release phase and document these stories in an iteration plan. This practice is done with the help of developers during these phases [22].

There is a general rule for agile processes that they are dealing with the most probable risks of project during the first release and primary iteration of each release [22]. However, there is no specific discipline in XP for analyzing the risks in requirements. Moreover, XP does not support formal review meetings.

FDA auditors expect a clear mechanism for evaluating requirements which is distinguishable based on existing products and documentation. However, XP does not mandate the presence of any documentation.

4.1.1.3. Mapping FDA Requirements Traceability Analysis to XP

FDA expects two traceability matrices at this stage to show the relation between system requirements and software requirements as well as system requirements and the result of risk analysis. However, there is no practice in XP to support the development of traceability matrices.

4.1.1.4. Mapping the FDA Test Plan to XP

XP requires from customers to explain the acceptance criteria of software before each release [22]. These acceptance tests could be executed by a customer or a developer. FDA requires an acceptance test plan before starting implementing story cards. In XP, the acceptance test plan which is known as the customer test document is defined by the customer [22]. Then these tests are automated by testers to generate acceptance tests.

System testing is supported in XP at the end of each release. The XP programmer (who plays the role of an architect) is responsible for performing system testing [18]. A System test plan is written by an architect and put in practice with the help of an XP tester.

4.1.1.5. Mapping FDA Configuration Management to XP

As discussed earlier, a version control is needed in a process as the software configuration management activity. Despite the existence of collective code ownership and continuous integration in XP there is no specific guideline in XP to support the use of version control [42].

4.1.2 Mapping to FDA Design

The software design in XP is done during the "iterations to release" phase by pair programming. The design in XP should be simple as the XP value. The architecture as the high level design is designed and built during first release in the "iterations to release" phase. Then this design is detailed more based on selected story cards to reach the implementation phase [13]. In addition, there is no documentation in XP that represents a software design specification. The design sketches are recoded informally by methods such as taking photo of design on whiteboard [22]. There is the necessity in XP to follow coding standard during "iterations to release". The coding standard is defined by chief programmer in a separate document.

4.1.2.1. Mapping FDA Design for Human Factors Engineering to XP

In XP, the user interface design (UI) is done during the "iterations to release" phase. In addition, due to the fact that XP tends to be a user-centered process by working with users throughout the process to obtain constant feedback and that it favours communication with customers, one can assume that XP considers the usable aspect of the final product although at a limited extent [39].

During the exploration phase, there is no specific practice in XP that mandates the use of usability design patterns or evaluate usability at the design or architectural level. To design the software system architecture, XP suggests building a system prototype during the exploration phase to evaluate different architectures. The final architecture is consolidated during the first release [13, 18]. There are two practices in XP that affect the design of the system architecture: System metaphors and simple design. System

metaphors are shared stories that describe how the system works and the simple design principle aims to make easier to understand each design component [48]. Despite the existence of the XP interaction designer, there is no explicit guideline in XP with respect to following specific architectural patterns and assessing the usability of the system at the architectural level.

4.1.2.2. Mapping FDA Software Design Evaluation to XP

The design in XP is kept as simple as possible and is informally documented. XP does not account for activities that deal with analysis of communication links among the system interfaces, control flow, and data flow as required by FDA.

Also, the design review in XP is significantly different from the FDA design review, because there is no formal design document in XP to review. Instead of having formal meetings, design review is limited to pair programming. It is recognized that an iterative cycle of pair programming provides continuous analysis and review of the design to improve and simplify it [13]. The development teams using pair programming have reported good quality of design with fewer lines of code as they worked together on both the design and the implementation [52]. In addition, using refactoring techniques, which are recommended in XP, improves the quality of design by making it extensible, modular, reusable, and maintainable [53]. However, XP does not support any specific practices which are required by the FDA for design verification and validation.

4.1.2.3. Mapping FDA Design Tractability Analysis to XP

There is overlap between design traceability analysis and design verification. Both of them put an emphasis on conformance of design with the requirements while traceability aims only to show the relations between the requirements and the design. XP does not support any traceability matrix during the design to show this relation.

4.1.2.4. Mapping FDA Update Test Plan to XP

Both unit testing and integration testing are performed within iteration in XP. But there is no discussion in XP about having test plan. The nature of testing in XP is explained in testing section.

4.1.2.5. Mapping FDA Test Design Generation to XP

The XP follows test driven development. Therefore, whole unit tests cases are generated after design in iteration. In addition, acceptance tests which are already generated by customer before iteration are automated by tester to execute those acceptance test cases at the end of iteration. Furthermore, Integration tests executed by XP programmer (integrator) at the end of iteration before acceptance test to integrate the pieces of code developed in iteration with produced software product [13, 22].

System test is done per release and supported by XP programmer (architect). As XP architect is the person who designed architecture, he is responsible for developing system test cases during multiple iterations [18]. Therefore XP supports developing test cases before coding as requested by FDA. However, there is no demand to have documentation for test procedure in XP.

4.1.2.6. Mapping FDA Configuration Management to XP

XP does not support version control as well as there is no SDS document to operate based on that. Despite the need of version control in XP activities such as collective code ownership and continuous integration, it is not covered in this process.

4.1.2.7. Mapping Update Risk Analysis to XP

The XP does not propose any discipline to handle design risk analysis. It just mentions that the high risk story cards should be implemented in preliminary iterations of each release [22]. In addition, using practices such as pair programming and refactoring can minimize the risk of having wrong design by improving design quality.

4.1.3 Mapping to FDA Coding and Construction

The coding in XP is performed iteratively using pair programming until a release is reached. The issue of selecting tools which are required by the FDA is not discussed in XP.

4.1.3.1. Mapping FDA Source Code Evaluation to XP

XP does not support any of the formal desk checking practices required by FDA. Instead, XP claims that pair programming is more successful than any inspection and formal review methods. According to experimental studies, the pair programming technique has been shown to be effective in uncovering errors in the code while programming, saving costs since errors are discovered before compilation [52]. Therefore, it is reasonable to

assume that pair programming satisfies FDA requirement with respect to source code evaluation without the need of having formal desk checking techniques.

4.1.3.2. Mapping FDA Code Documentation to XP

XP does not support the production of any documentation needed by FDA such as documenting modules, errors, the compilation process, and the used tools and techniques.

4.1.3.3. Mapping FDA Code Traceability Analysis to XP

XP does not support the development of traceability matrices from source code-to-design, from test cases-to-source code, from test cases-to-design, from test cases-to-risk analysis, and from source code-to-risk analysis as required by FDA.

4.1.3.4. Mapping FDA Source code interface analysis to XP

There is no specific guideline in XP on how to analyze different interfaces of the subsystems for implementing and integrating the various parts of the code.

4.1.3.5. Mapping FDA Test Generation to XP

There are two types of test cases in XP that are possible to map to the design: unit and integration test cases. XP follows the test-driven development method (TDD), in which test cases are designed before coding starts. Therefore, unit test cases are generated as the result of TDD after the design and before coding. Integration test cases are generated at the end of an iteration once a new piece of code is adding to the collective codebase. These two test cases have the potential to map to the code and the design. For the acceptance test, it is not possible to map it to elements of design or code. Thus,

acceptance test cases do not include to map design and code. In addition, dividing XP into small iteration and having simple design facilitate developers to make relation between the elements of design and code to develop code to design traceability matrix. System test cases are generated during the whole release to be done on productionizing phase. As for acceptance tests, they are defined by customers and generated in a form of automated test cases by XP testers. They are executed at the end of a release.

4.1.4 Mapping to FDA Testing

4.1.4.1. Mapping FDA test documentation to XP

Except a limited number of documents that are created and necessary to reach a running product, XP attempts to minimize the amount of efforts on documentation as one of its values. Among the testing documents created in XP, the test cases exist usually in a form that is readable by automatic testing tools.

4.1.4.2. Mapping FDA test execution to XP

XP supports unit testing. In each iteration unit tests are generated based on the design to test subsequent code. Then, after writing the code, the unit tests are executed to find probable fault in software [54].

In addition, XP supports integration testing by continues integration practice. The integration testing in XP is different from traditional integration test. In the traditional way, integrating code into product is conducted for full volume test [54]. However in XP, there is continuous integration whenever new code is written, added to the collective codebase, and unit tested [13]. In other words, instead of spending time to integrate the

whole system throughout the project and finding compatibility errors, integration is done based on new code submitted by developers within iteration [49]. The collective codebase serves as the code repository that responsible for integrating new or updated codes by running all previous unit tests. Also in XP, a complete regression test bed is provided by continuously running unit tests to integrate new code into collective codebase [54]. Also, the previous acceptance test cases are employed as regression tests before deploying the release [49].

There is system test in XP with the scope limited to architecture. An XP programmer (architect sub-role) is responsible for providing the system tests to examine the architecture during the productionizing phase for each release [18].

Beck et al. believes that the XP programmer (architect) has to provide system test cases to be able to examine the system architecture [18]. We took that idea to have system test per each release during the productionizing phase.

4.1.4.3. Mapping FDA Test traceability Analysis to XP

There is no support in XP for traceability matrices from unit tests to detailed design, from integration tests to high-level design, and from system tests to software requirements. But developing such traceability matrix for unit test to detailed design should be straightforward since the unit tests are developed based on the design and before coding.

4.2 Summary

In this chapter, we analyzed the capability of XP as our intended software development methodology to support FDA medical devices' requirements on software processes. The

result of this analysis is summarized in Table 4-1. XP lacks support for many FDA process requirements. Thus, for developers who want to use XP to build medical device software, there is an important need of extending it based on FDA medical device process requirements.

Table 4-1. Level of XP support for Medical Devices' Process Requirements

Development phases	Validation Sub-phases	XP Support for Typical Practices	XP Support for Typical Documentations
Requirements	Requirements Elicitation	User story card writing High customer involvement Eliciting requirements in number of iterations	User stories are the only documentation of requirements in XP
	Requirements Evaluation	System prototype Building software functionality in number of iterations Handling most probable risks early in development	The process of evaluating requirements is not documented in XP
	Requirements Traceability Analysis	There is no practice in XP for traceability analysis	There is no documentation that relates different artifacts
	Test plan	Customers are involved in the writing of acceptance tests The XP architect is responsible for creating a system test plan	Both system and acceptance test plans are documente
	Configuration Management	No guideline	No documentation
Design	Design for Human Factors Engineering	System prototyping to obtain feedback for the end users, Not enough support	The prototype itself is the only evidence that prototyping took place
	Software Design Evaluation	Pair programming Refactoring	Design evaluation is not documented
	Design Traceability Analysis	There is no practice in XP for traceability analysis	There is no document that relates different artifacts
	Update Test Plan	Updating test plans taking into account design elements	Unit and integration test plans
	Test Design Generation	Generating test cases for unit, integration, acceptance and system testing.	Unit test case Integration test case Acceptance test case System test case
	Configuration Management	No guideline	No documentation
	Update risk analysis	No guideline	No documentation
Coding and Construction	Source Code Evaluation	Pair programming	No documentation is produced
	Source Code Documentation Evaluation	There is no support for this activity	Since the code does not need to be documented, the evaluation of the documentation does not apply
	Code Traceability Analysis	There is no practice in XP for traceability analysis	There is no documentation that relates different artifacts
	Source code Interface Analysis	There is no support for this activity	No documentation is created
	Test Generation	Updating test cases for unit, integration, acceptance and system testing.	Unit test case Integration test case Acceptance test case System test case
Testing	Test Documentation	XP is a test-driven approach and there are many practices and roles that are dedicated to testing.	Test plans, test procedures, test logs, and test cases
	Test Execution	Unit testing Integration testing Acceptance testing System testing	Test logs are kept for debugging purposes

Chapter 5 - Extension of XP Process

In this chapter, we propose an extension to XP, which we model using the Eclipse Process Framework (EPF). The extension of XP has been evaluated by XP experts.

5.1 Extension Methodology

As defined in section 2.3, a software process is defined as a set of activities, methods, practices, and transformations that are used to develop and maintain software and its associated products [12]. An activity in process obtains the input artefacts and transforms them to output artefacts. Each activity itself can be divided into a number of tasks and practices (e.g. writing different design diagrams as the design tasks). Each process defines the required skills in a form of different roles to assign them activities and practices according to their expertise (e.g. architect, analyst) [55]. In addition to those process elements, the knowledge and methods about how to do different process activities and practices are encapsulated in processes description (e.g. object oriented design) [55].

Our extension mechanism is therefore based on adding new roles, practices and artefacts to XP throughout its various activities to meet the FDA requirements.

5.2 Extending XP to Meet FDA

In this section, we elaborate the analysis behind the proposed extension of XP. The extended XP is explained according to the FDA requirements discussed in Chapter 3.

5.2.1 Requirement Phase

5.2.1.1. Requirement Elicitation

To meet FDA regulations with respect to the requirement elicitation phase, we need to extend XP so as to support documenting the requirements in the form of a Software Requirement Specification (SRS) document as required by the FDA. We therefore suggest extending XP to support the SRS document. Our extension also involves adding new sub-roles to XP as well as adding new practices.

The customer writes stories about the system's features that are expected to be available for the first release. An XP programmer (more precisely the interaction designer) leads the customer in this process through specific questions (e.g., "is the story testable?"). The functional and non-functional requirements are expressed in user stories with respect to user characteristics and other HFE requirements. We also suggest that the XP interaction designer puts more emphasis on elucidating the user characteristic of the system during the writing of stories.

On the other hand, the lack of support for non-functional requirements (NFR) is reported as one of the common deficiencies of XP projects [50]. However, defining NFRs using story cards and software requirement specification document should be sufficient to meet

the FDA requirement. In that respect, we propose that the XP interaction designers and the XP testers can guide customer through the definition of the NFRs.

As mentioned before, XP does not have the needed activities and guidelines to support the production of software inputs, software outputs, and interface requirements. To address this, we propose that the XP interaction designer and XP tester derive customers to provide the sufficient level of quality expected from the product such as safety and reliability. XP advocates daily measurement of quality based on defined metrics such as success rate of running tests [22]. It is the responsibility of the XP tracker to collect the metrics and measure the progress of the defined quality attributes. Thus, the XP interaction designer (sub-role of programmer), tester, and tracker are three important roles to support the definition and the measurement of quality attributes.

In addition, the external interface requirements should be clearly identified by the developers with the help of domain experts. We suggest that the XP architect (sub-role of programmer) to be the one responsible for defining and documenting interface requirements since the interface requirements are used to define the structure (architecture) of the system.

The process of requirement elicitation in XP is done during the exploration, planning, and "iterations to release" phases. The stories are written in the exploration and planning phases and then in the "iteration to release" phase, each story is decomposed to measurable tasks assigned to developers. The roles of the XP tracker, XP programmer (interaction designer, architect) and the XP tester are therefore very important in the elicitation of the requirement. We suggest assigning the role of producing an SRS to the

programmer (architect) to manage the SRS documentation based on requirements inputs and requirements outputs.

The progress of eliciting the functional and non-functional requirements is tracked by the XP tracker throughout the development process. Besides, the elicited requirements are tested based on acceptance tests (customer tests) and system tests. The automation and the execution of these tests are the responsibility of the tester in XP. In Addition to system tests written by an XP programmer (architect sub-role) to stress the architecture of system, we suggest to have the XP programmer (the interaction designer sub-role) and the XP tracker to write additional systems test based on their performance and other quality metrics.

Figures 5-1, 5-2, and 5-3 show the new practise and artefacts (in red color) that need to be added to XP to support the requirement elicitation phase as per FDA requirements for device software.

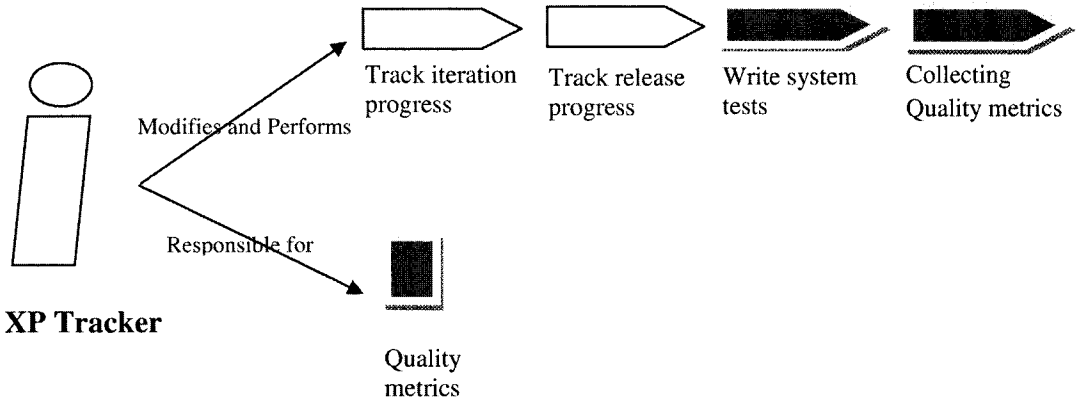

Figure 5-1. The XP tracker new practices and artefacts

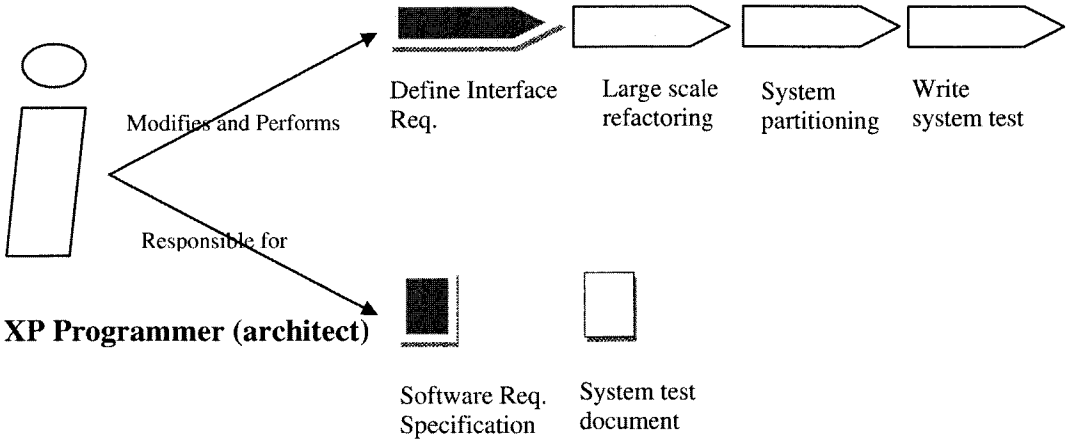

Figure 5-2. The XP architect new practices and artefacts

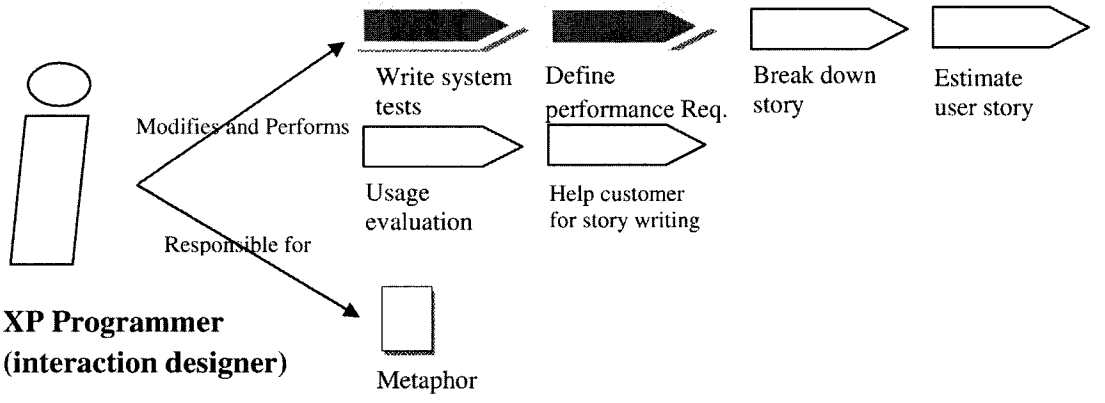

Figure 5-3. The XP interaction designer new practices

5.2.1.2. Requirements Evaluation

The XP does not support formal review meetings as requested by the FDA. To address this, we suggest adding review meetings during the "iterations to release" phases. The written user stories and SRS can be evaluated at the end of each iteration and release. In these review meetings, the XP programmers should present their work to customers and other stakeholders.

The XP does not support risk analysis in the requirement phase as required by the FDA. However, adapting a risk management strategy to a specific software process relies on the scale of the project and the probable risks that may happen in future. The agile processes like XP claim that they are risk driven [56]. They favour dealing with risks early in the process with the help of techniques such as requirement prioritization, but they do not propose a stringent discipline to overcome the risks if they occur during the software development cycle. Rather, XP leaves this up to project managers and developers to choose and adapt their own risk management strategy based on the importance of the project. Since the FDA does not prescribe any specific risk management techniques then we consider the actual way of handling risks in XP sufficient to meet the minimum acceptable risk analysis from the FDA's point of view.

5.2.1.3. Requirement Traceability Analysis

FDA requires supporting traceability analysis within development. In Figure 5-4, the

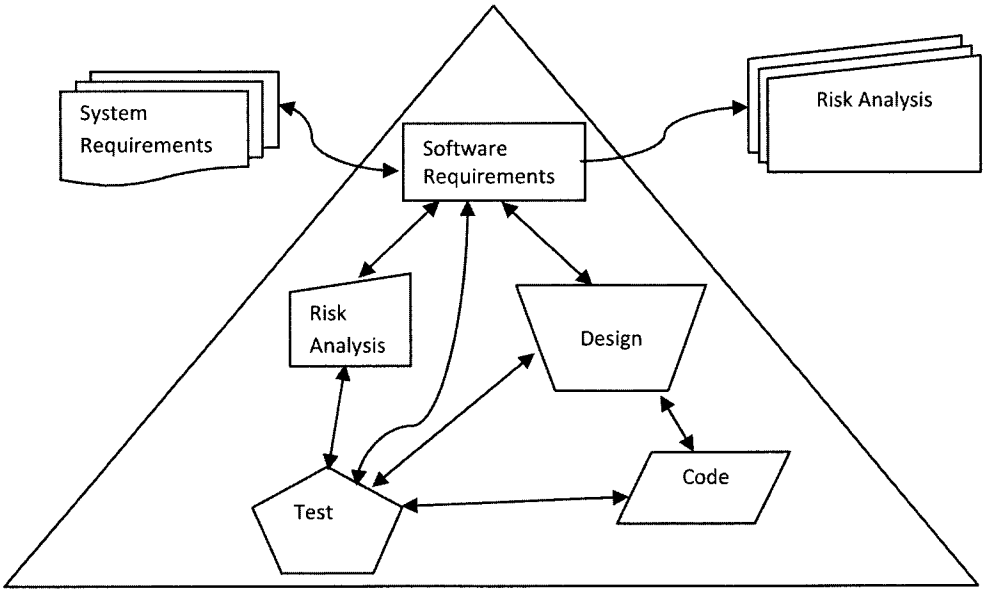

Figure 5-4. FDA Traceability relations in software development

triangle distinguishes the scope of the traceability relations among various process activities.

Adding support for traceability to XP may decrease in the agility of the process, if done and managed manually, since many documents need to be produced to connect various process artefacts to each other. To address this, we propose to rely on CASE tools as much as possible. In the requirement phase, the FDA requests to provide two traceability matrices to connect software requirements to system requirements (and vice versa), and software requirements to the results of risk analysis.

To relate software requirements to system requirement, we propose to create a traceability matrix that connects the SRS to the documents that contain computer-based system requirements and which are defined during the project feasibility study. Moreover, there should be a unidirectional traceability matrix to show the relation between software requirements and the risk analysis results. An example of traceability matrix is presented below:

Requirement Source	Product Requirements	HLD Section #	LLD Section #	Code Unit	UTS Case #	STS Case #	User Manual
Business Rule #1	R00120 Credit Card Types	4.1 Parse Mag Strip	4.1.1 Read Card Type	Read_Card _Type.c Read_Card _Type.h	UT 4.1.032 UT 4.1.033 UT 4.1.038 UT 4.1.043	ST 120.020 ST 120.021 ST 120.022	Section 12
			4.1.2 Verify Card Type	Ver_Card _Type.c Ver_Card _Type.h Ver_Card _Types. dat	UT 4.2.012 UT 4.2.013 UT 4.2.016 UT 4.2.031 UT 4.2.045	ST 120.035 ST 120.036 ST 120.037 ST 120.037	Section 12
Use Case #132 step 6	R00230 Read Gas Flow	7.2.2 Gas Flow Meter Interface	7.2.2 Read Gas Flow Indicator	Read_Gas _Flow.c	UT 7.2.043 UT 7.2.044	ST 230.002 ST 230.003	Section 21.1.2
	R00231 Calculate Gas Price	7.3 Calculate Gas price	7.3 Calculate Gas price	Cal_Gas_ Price.c	UT 7.3.005 UT 7.3.006 UT 7.3.007	ST 231.001 ST 231.002 ST 231.003	Section 21.1.3

Figure 5-5. Example of traceability matrix (taken from [57])

The above traceability matrix is bidirectional. In other words, both forward and backward traceability is possible in that matrix. Thus, it is possible to show software/system requirements traceability with one matrix.

We suggest having the XP architect (a sub-role of the XP programmer) role to define the relation between the system requirements and software requirements as well as provide documentation (Figure 5-6). The establishment of this relation should be enabled using a CASE tool to automate the management of the traceability matrix. Similarly, we assign the task to relate requirements to risk analysis and produce the similarity matrix to the XP architect. These traceability matrices should be defined at the time of eliciting the requirements during the exploration, planning, and "iterations to release" phases [22].

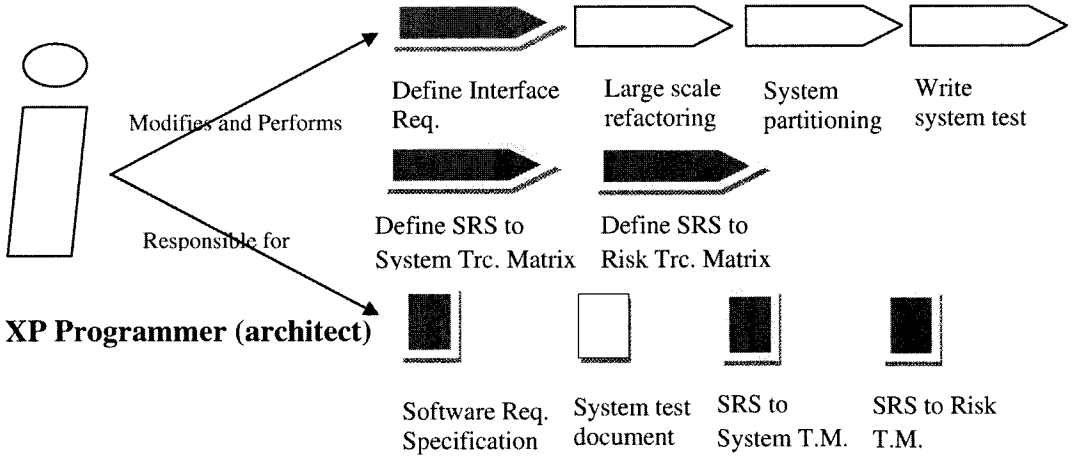

Figure 5-6. The XP architect new practices and artefacts to support traceability during requirement analysis

5.2.1.4. Test Plan

XP supports the acceptance test plan as a customer test. In our extension, we considered the XP tracker and interaction designer for writing system tests in addition to the XP

architect (programmer). However, the XP architect is the main person responsible for writing system test plan, partitioning system test among other roles, and ask them for required documentation. Writing system test plan has to be done at planning phase or at the beginning of iterations to release phase.

5.2.1.5. Configuration Management and Software Requirements Specification

Software configuration management (SCM) composed of a set of activities to control change within software development process [41]. This set of activities should be established in development process to identify the changes, control and implement the changes, and auditing and documenting the changes [41].

The FDA demands two activities for managing configurations: traceability analysis and version control. Traceability analysis is described as a separate section (for each process phase) in this thesis. Version control systems largely rely on CASE tools and therefore there is no need to add any practice or role to XP to support configuration management tasks.

5.2.2 Design

5.2.2.1. Design for Human Factors Engineering

We propose an extension to XP to take into account usability requirements during the system design, as required by the FDA. We achieve this by proposing to divide the exploration phase into sub-phases that enables a better user-centered design. This is based on the work of Obendorf et al. [56], where the authors defined sub phases of the exploration phase to meet usability in design [58]. As illustrated in Figure 5-7, the

exploration phase contains following sub-phases before prioritizing stories for each iteration: contextual investigation, requirements scenario, vision/use scenario and stories.

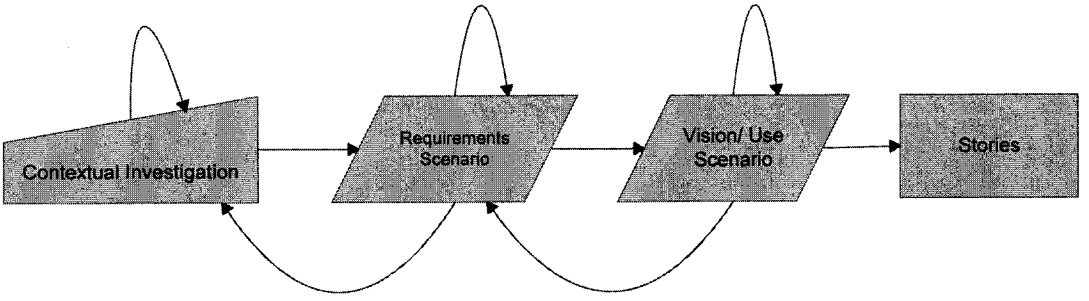

Figure 5-7. The Extended XP Exploration Phase (based on [56])

Contextual investigation is the method for understanding the application domain. It is based on interviews and workplace observations to understand the usage context, responsibilities, and relationships of end users [59]. Requirement Scenario has its roots in the area of problem scenario in scenario-based engineering [60]. Requirement scenarios provide details about the functionality of the final product. The vision or use scenario phase provides a consistent system look with problem statement and its corresponding solution for the specified system [59]. Additional investigation and feedback from users in each increment enhances the requirement scenarios and vision. The extended XP life cycle is shown in Figure 5-8.

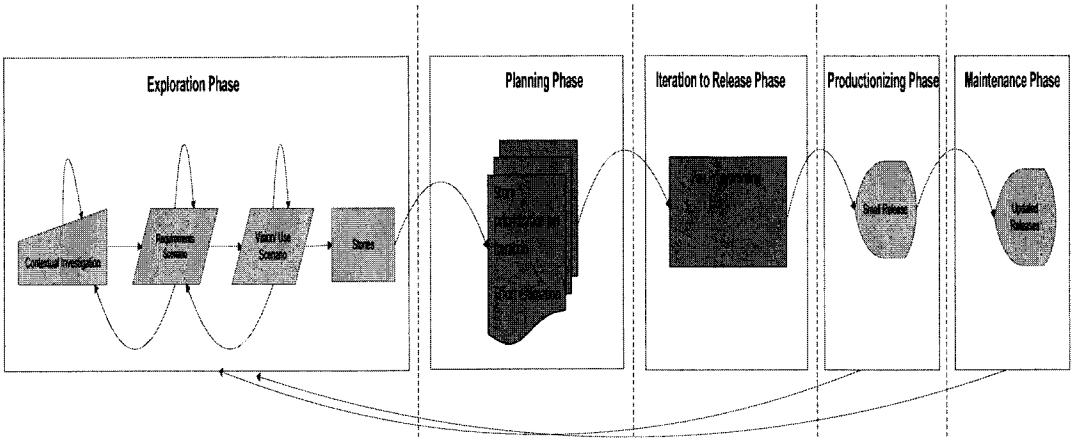

Figure 5-8. Extended XP life cycle

Besides extending the exploration phase in XP to handle usability during the design, the extended XP could contain practices such as prototyping and redesigning during the iterations to release phase [59]. As mentioned before, in the first iteration during a release, the structural stories are selected to consolidate the system architecture. Based on the feedbacks obtained from the user after each iteration, the design could be refined or redesigned according to the significance of feedbacks. Since XP design models are informal like drawing on whiteboard, we suggest recording all informal design models that can later be used for refining the design. In addition, as we discussed in Section 4.1.2, using prototyping techniques during design increases the chances of developing usable software. Therefore, by doing paper prototyping after the design during the "iterations to release" phase, the design model could be evaluated for its usability. The extended "iterations to release" phase is shown in Figure 5-9.

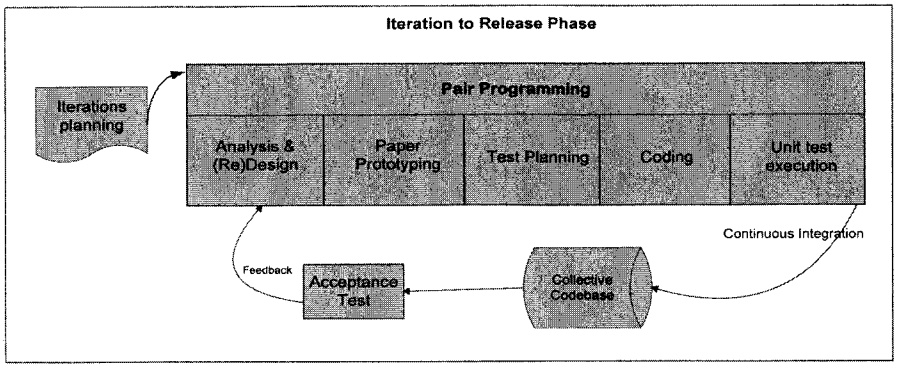

Figure 5-9. The Extended XP Iterations to Release Phase

We suggest having the XP architect be responsible to ensure that usability is taken into account during the exploration phase (Figure 5-10). The XP architect can achieve this by working with the customers. In the "iteration to release" phase, we suggest to have the XP interaction designer be responsible of paper prototyping (Figure 5-11).

The new extended practices for these new roles are showed below:

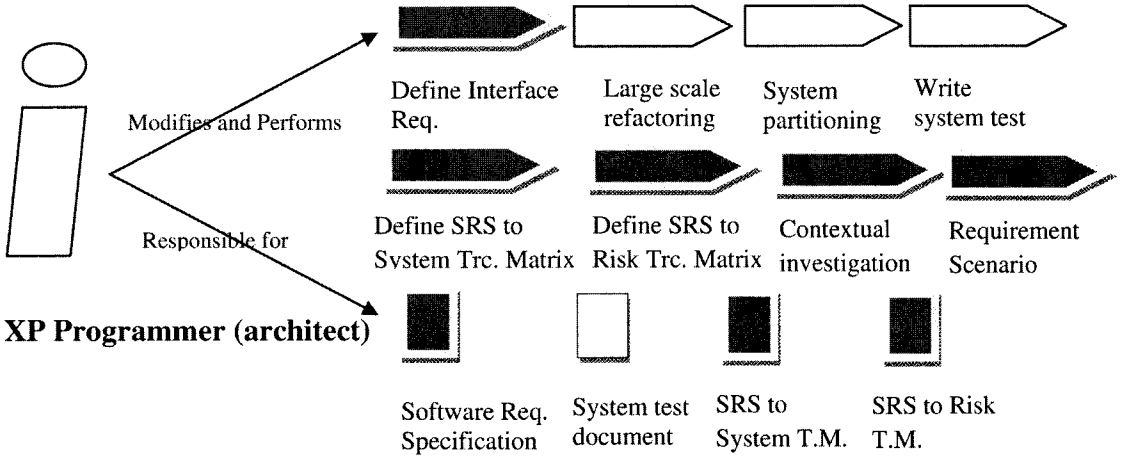

Figure 5-10. XP architect

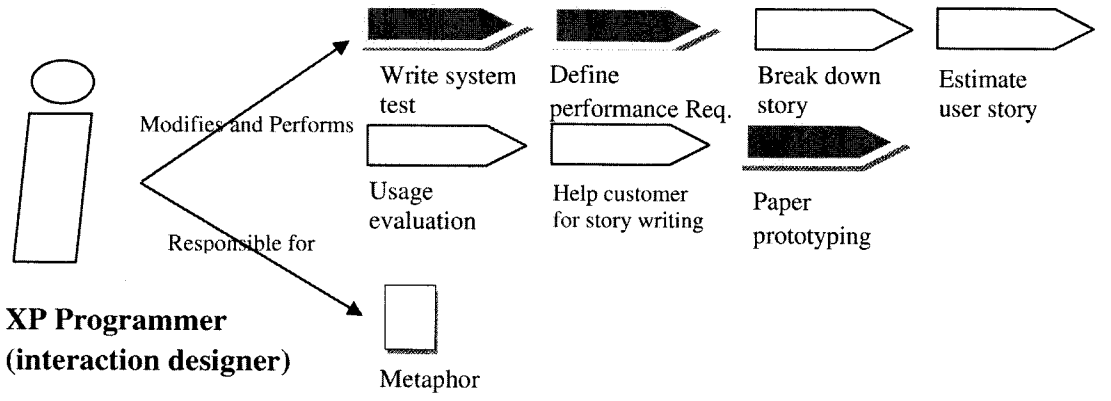

Figure 5-11. XP interaction designer

5.2.2.2. Design Evaluation

As discussed in Section 3.2, prototyping is considered as a method for design evaluation. We do not recommend separating different practices for design review, design verification, and design validation as the FDA suggests. Because, these three parts of a design evaluation have considerable overlaps with each other and to avoid having a heavy process. However, informal and undocumented design as XP suggests is also not a good case for design in large-scale projects. We suggest considering paper prototyping, discussed earlier in the context of usability, as the design verification practice.

5.2.2.3. Design Traceability Analysis

In the design phase, the FDA requires a bidirectional traceability matrix between the design specification and software requirements. XP suggests a number of artefacts during the design phase such as a metaphor, CRC cards, and the iteration and release plan. These documents can be seen as a software design specification that can be traced to the

software requirement specification to provide the requirement/design traceability matrix during the planning and the "iteration to release" phases (Figure 5-12). This traceability matrix has to be managed using CASE tool to keep the process light.

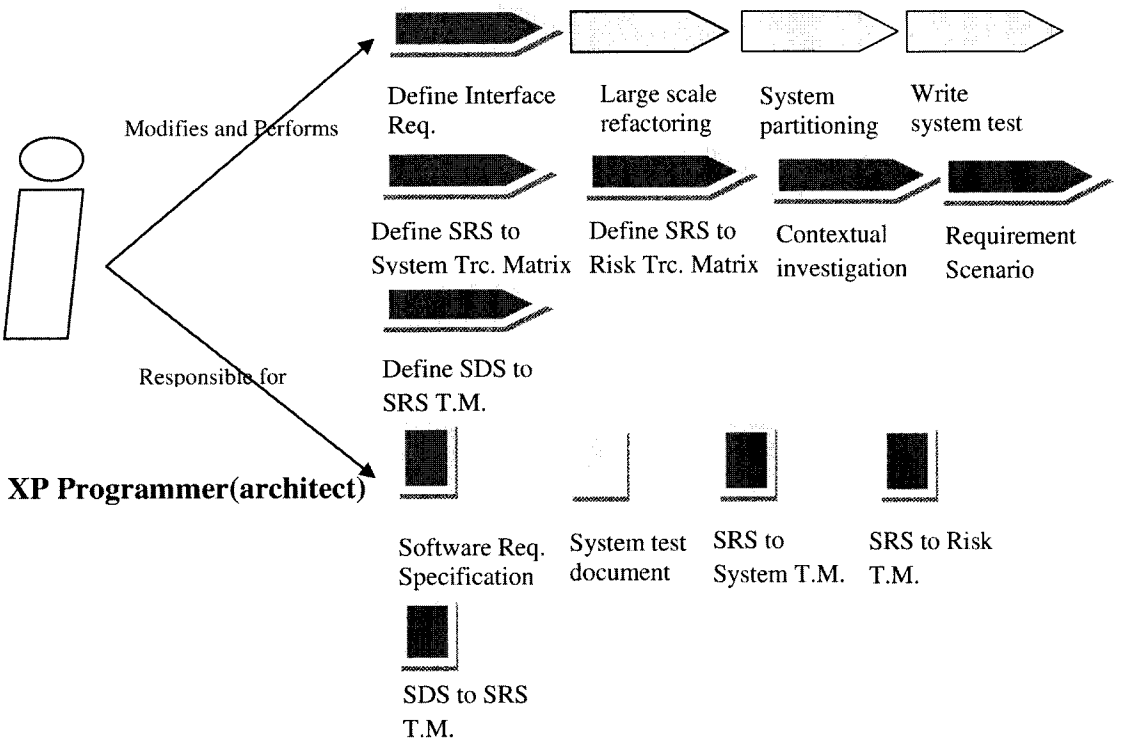

Figure 5-12. The XP architect new practices and artefacts

5.2.2.4. Updating Test Plan

We suggest to have the chief programmer in XP that assigns tasks to progrmmers write unit test plan (Figure 5-14). In addition, we propose the XP integrator (sub-role of programmer) to provide the integration test plan document (Figure 5-13). These two aditional documents are shown according to their role below.

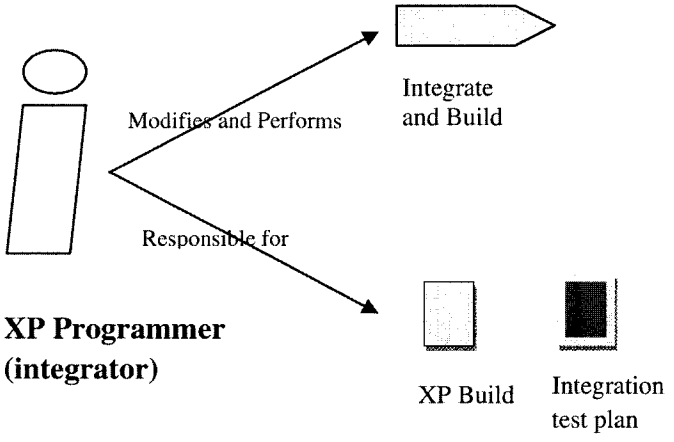

Figure 5-13. XP integrator

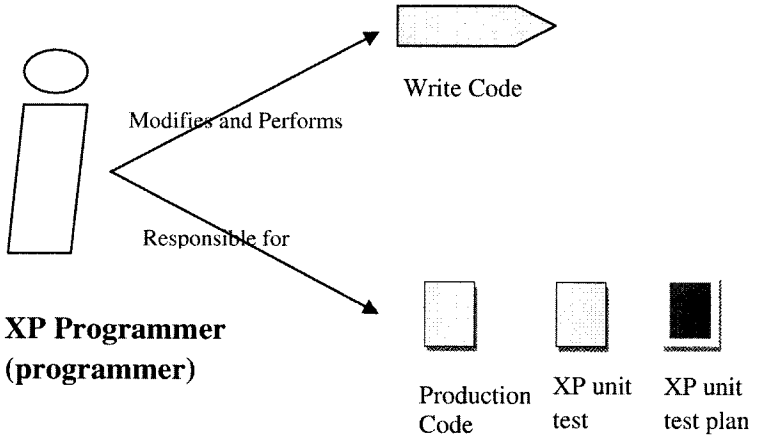

Figure 5-14. XP programmer

5.2.2.5. Test Design Generation

As explained in Section 4.2, XP supports creating all test cases before coding starts. But there is no practice to provide test procedure in XP. We do not suggest to document test procedures as the test cases are already produced. It is important to mention that we suggested to do system test per each iteration in our extension to augment the level of testing of the software. Thus, the XP architect has to provide and automate test cases per each iteration in addition to the ones applied to the entire release.

5.2.2.6. Software Configuration Management

Due to multiple iterations and releases, design is subject to change and we need to use the specified version control system to keep track of the changes. The version control system is usually selected regarding the documentation policy (documenting requirements and design in our case) used in the project. We follow XP policy to leave it up to process managers of the project to choose a particular version control system to process managers. We only discuss the mandatory practices that version control system has to fullfil. We suggest to have the XP architect (programmer) be the one reponsible for documenting the design as software design specification (SDS) and update the version control system with the new SDS (Figure 5-15). He or she can assign the documentation of lower-level design to the XP inteaction designer during the development phase.

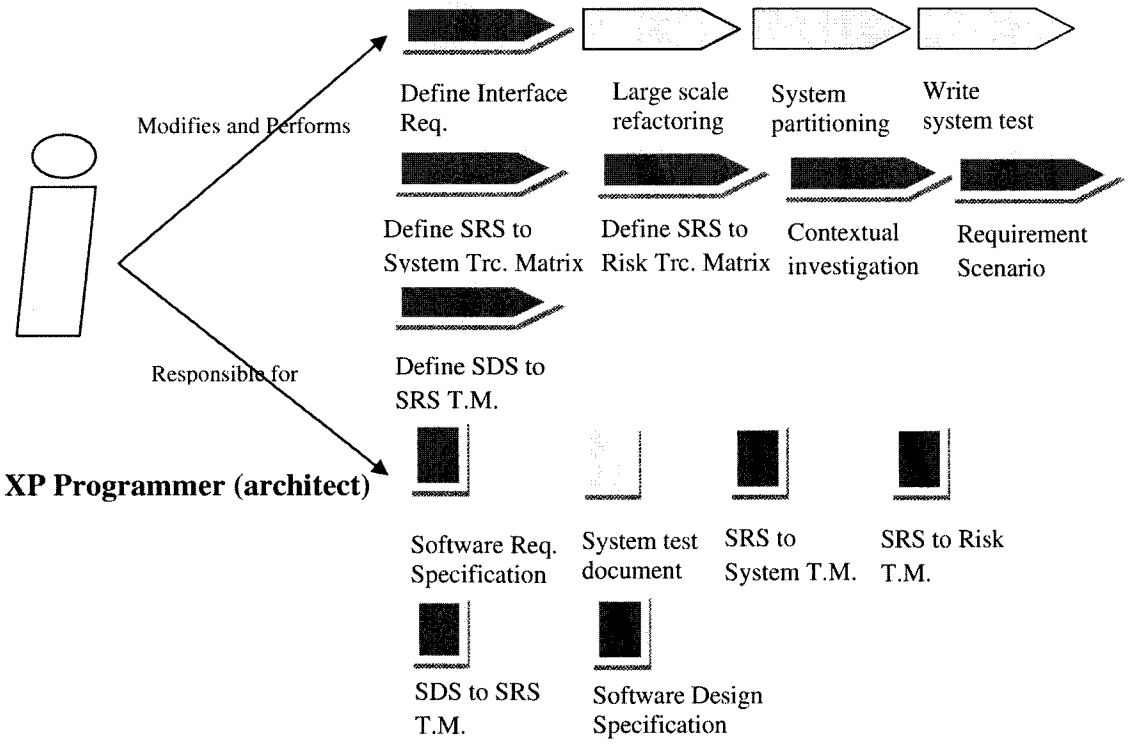

Figure 5-15. XP architect

5.2.3 Coding and Construction

5.2.3.1. Source Code Evaluation

As mentioned in Section 4.2, there is no need to add any activity to XP as pair programming supports source code evaluation.

5.2.3.2. Source Code Documentation Evaluation

We do not suggest having heavy practices for documenting the source code. We suggest to rely on CASE tools that automaticly generate documentation for code (based on comments, naming conventions, etc.).

5.2.3.3. Source Code Traceability Analysis

In coding phase, FDA requires traceability matrices between:

Design specification ⟷ Source code

Test Cases → Source code

Test Cases → Design Specification

Test Cases → Risk Analysis Results

Source Code → Risk Analysis Result

We assign the tasks of producing these matrices to the XP architect who has a good overview of the entire system and who oversees the work of other XP programmers. However, it is expected from the XP architect to work in collaboration with the XP tester. Also, there should be tool support to automate the management of these matrices.

5.2.3.4. Source Code Interface Analysis

During pair programming and refactoring, the XP interaction designer can analyze and evaluate the interfaces among the various interfaces of the system (Figure 5-16). We, therefore, assign analyzing source code interfaces to the interaction designer during pair programming.

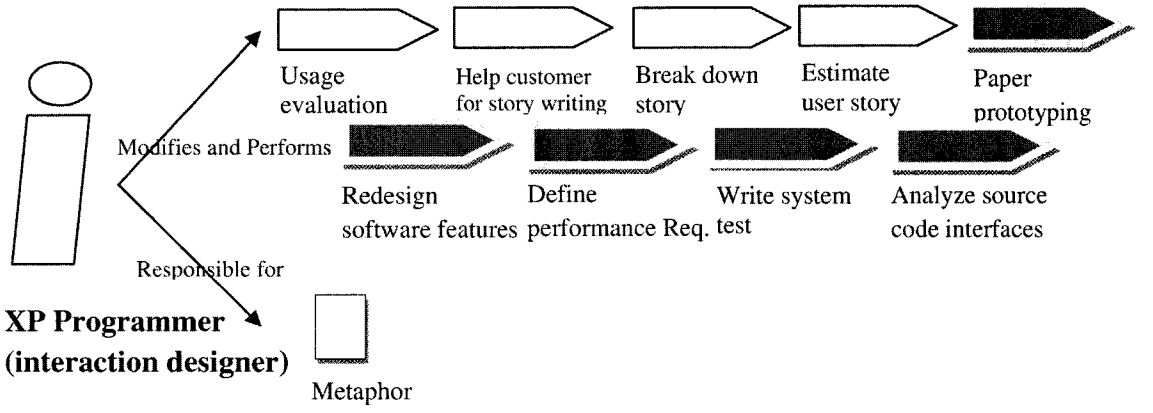

Figure 5-16. XP interaction designer

5.2.3.5. Test Generation

As pointed out in Section 4.2, XP generating unit, integration, and acceptance test before and along coding activities. As in this extension system test is done by per iteration, the test cases have to be designed and generated by the XP architect (programmer) when coding is done during an iteration.

5.2.4 Testing

5.2.4.1. Test Documentation and Execution

XP has already practices that permit the generation and the execution of the required test plans and test cases. Therefore, no extension is needed at this stage.

5.2.4.3. Test Traceability Analysis

In test phase, FDA requires the following traceability matrices:

Unit test ⟶ Low level design

Integration test ⟶ High level design

System test ⟶ Software requirements

Similar to Section 5.2.3.3, we assign the tasks of producing these matrices to the following programmer sub-roles: We assign system-to-software requirement traceability matrix to XP architect who is responsible for SRS. In addition, integration test-to-high level design traceability matrix is assigned to XP integrator as he is the one who is responsible for integration test. Finally we assign unit test-to-low level design to XP interaction designer as he is dealing with lower level design in XP.

5.3 Perform the Extension to XP and model it by EPF

The extension of XP to satisfy FDA medical device software has been implemented using the EPF (Eclipse Process Framework) as a process model library to be used by developers. A summary of the newly added practices and artefacts are shown in Table 5-

1. In addition, the entire process life cycle of the extended XP is illustrated in Figure 5-17.

Table 5-1. The extended XP practices and work products

Role	Practice	Artieact
XP Coach	Adapt & improve process	
	Explain process	
	Improve team skills	
	Keep process on track	
	Resolve conflicts	
XP Customer	Adjust iteration scope	User story
	Define customer test	XP customer test
	Define iteration	XP iteration plan
	Define release	XP release plan
	Report customer test result	XP vision
	Report project status	**Use senario**
	Revise release plan	
	Write user story	
XP programmer	Break down story	Coding standard
	Define coding standard	Metaphor
	Estimate task	Production code
	Estimate user story	XP build
	Implement spike	XP unit test
	Integrate & build	System test for architecture
	Refractor code	**SRS**
	Help customer for story writing	**SDS**
	Usage Evaluation	**Traceability matrices**
	Large scale refactoring	**Integration test plan**
	System partitioning	**Unit test plan**
	Write system test for architectur	**System test plan**
	Write code	
	Define interface req.	
	Define performance req.	
	Define traceability matrices	
	Contextual investigation	
	Requirement Scenario	
	Redesign software feature	
	Paper prototyping	
XP programmer(administrator	Setup programmer environment	
XP tester	Automate customer test	
	Run customer test	
	Setup tester environment	
	Part of automating system test	
XP tracker	Track iteration progress	**Quality metrics**
	Track release progress	
	Writing system test	
	Collecting quality metrics	

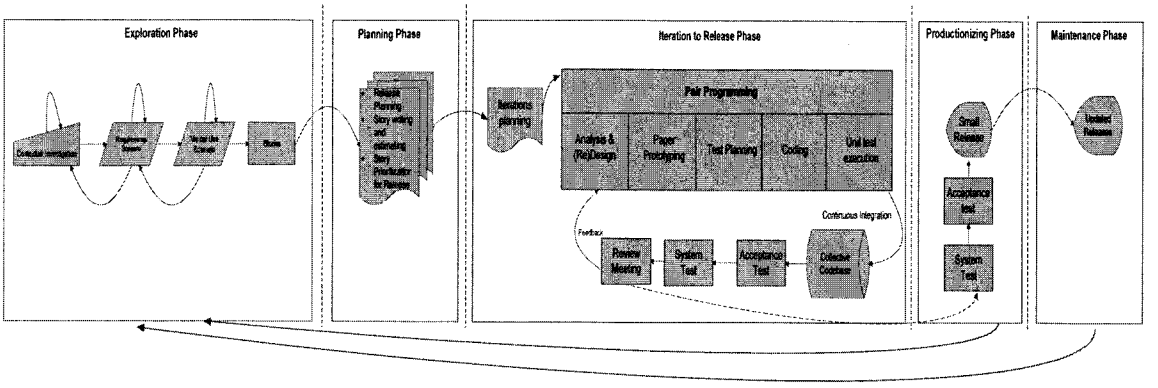

Figure 5-17. The extended XP life cycle

5.4 Evaluation

In the area of software process improvement, there are two methods of evaluating the improved process. The best way to evaluate the improved process is by applying it in real project, and tracking the progress of the work. However, it is difficult to find the development team to accept changing their development process due to inadvertent cost that might be added to project. The second approach, which we used in this thesis, of evaluating an improved process is based on a questionnaire. We designed a questionnaire that contains five questions and asked two XP experts to fill out the questionnaire. The questions are designed to evaluate whether the extended XP keeps the values of the traditional XP. The compliance of the new process with FDA has not been a concern since it has been achieved systematically by mapping every FDA requirement to XP. The issue is that by doing so, we might have missed some of XP principles. The questionnaire is based on five-scale answers ranging from Excellent, Very Good, Good, Fair, to Poor.

The questions and the results of the evaluation are presented in what follows:

Q1. Considering the new practices and artefacts that have been added to the traditional XP, do you think that the proposed version of XP is sufficient for developing safe and reliable software?

The reviewers rate this question "good" and "very good" respectively. Both agreed that the new approach recommends using a tracking method (through traceability matrices) in order to track the progress and quality of work, which should ultimately result in a reliable and safe system.

Q2. How well the new process handles version control, traceability, and other documentation overhead to meet FDA requirement without impacting the agility of the process?

The participants' answers are "good" and "very good" respectively. They believe that the distribution of documentation should be handled in a way that the programmer is not overwhelmed with documentation that will distract him or her from completing the project/development work. In addition, one of the participants mentioned the point that time should be managed properly by the project manager and tasks should be distributed evenly.

Q3. Considering the fact that most of the required documentation and heavy weight practices required by FDA (SRS, SDS, traceability matrices, coding documentations, testing documentations) are handled in this extension by relying on CASE tools. Do you think that this is a good decision knowing the limitations of existing CASE tools?

The participants' answers are "fair" and "very good" with respect to the capability of extended XP to use CASE tools. The opinions of the participants vary, perhaps, due to

the experience of each participant with existing CASE tools. However, both participants agreed that some degree of automation for managing documents and traceability matrices is needed.

Q4. Do you think the reliance of the extended process on practices such as pair programming and refactoring can result in better software quality in terms of better design and less bugs (and therefore eligible to meet FDA)?

The participants' answers are "good" and "very good". They also added that these techniques can only be effective if the programmers are trained on these methods.

Q5. The FDA guidelines founded upon traditional and plan driven processes that require rigid practices for all aspects of development. On the other hand, agile processes are less disciplined. Considering FDA process requirements, how well is the work presented in this thesis in terms of mapping FDA requirements to XP?

The participants' answers are "good" and "very good". Both participants believe that it is not easy to extend a process such as XP to meet FDA requirements (which are based on heavy documentation) without compromising the agility aspect of XP. One reviewer added that the extension needs to be validated in practice.

The result of this preliminary evaluation shows that our extension of XP to meet FDA requirements for medical device software is promising. However, we are aware that we need further experiments involving many more users to be able to assess the effectiveness of our result. In addition, we also need to study how the extended XP is used in projects that involve FDA requirements.

Chapter 6 - Conclusion

6.1 Research Contributions Revisited

In this thesis, we discussed the FDA requirements for medical device software. We extracted the requirements that FDA imposes on software processes by focusing on process activity. Next, we analyzed the capability for XP (an agile process) to support FDA requirements. This analysis is not based on mere practices and documents, but it is also based on the certification techniques required by FDA. We uncovered areas where XP fails to support many of the key requirements. We proposed an extension to XP by adding new practices and artefacts. Our extension attempts to balance agility and the need for extended documentation as required by FDA.

6.2 Opportunities for Further Research

The immediate future work would be to experiment with the extended XP in practice. We are also aware that additional evaluations are needed involving more process experts. Another future direction would be to apply the techniques presented in this thesis to other agile processes such as Scrum, FDD, etc.

In addition, As FDA requires auditable software to show its quality, there are still many regulatory requirements in FDA which are beyond development process (e.g. risk management) and have to be handled throughout the entire organization (at the level of the business processes of the company). It would be interesting to invest into studying a

broader support of IT companies for FDA requirements that goes beyond a software process view.

6.3 Closing Remarks

Regulatory compliance has become a key challenge in today's ever-changing business environment. Software companies need to comply with many regulations and laws that apply to their client base. Perhaps, one of the most heavily regulated industries is the one regulated by the Food and Drug Administration (FDA). In this thesis, we studied the difficulties to comply with the FDA regulations for medical device software. We think that the extracted software process requirements from FDA guidelines and the modeled extension of XP by EPF give better insight about FDA software development requirements to software companies for passing FDA audit. In addition, the extended XP can be a good instrument to teach the benefits of XP in projects that required compliance with regulatory requirements. We acknowledge that there will be a need for tool support such as for the production of traceability matrices. Finally, we hope that the work presented in this thesis brings a significant contribution to the field of regulatory compliance and its impact on software development.

References

[1] I. Lee, G. J. Pappas, R. Cleaveland *et al.*, "High-Confidence Medical Device Software and Systems," *Computer-IEEE Computer Society-*, vol. 39, no. 4, pp. 33, 2006.

[2] FDA, "General Principles of Software Validation; Final Guidance for Industry and FDA Staff," H. a. H. Services, ed., 2002.

[3] J. Forsström, "Why Certification of Medical Software Would Be Useful?," *International journal of medical informatics,* vol. 47, no. 3, pp. 143-151, 1997.

[4] M. M. Abdeen, W. Kahl, and T. Maibaum, "FDA: Between Process and Product Evaluation."

[5] "Software and Systems Process Engineering Meta-Model Specification," http://www.omg.org/cgi-bin/doc?formal/2008-04-01.

[6] M. G. Silverman, *Compliance Management for Public, Private, or Non-Profit Organizations*: McGraw-Hill, 2008.

[7] K. Reilly. "Compliance Spending," URL: http://www.amrresearch.com/Content/View.asp?pmillid=19239.

[8] CNN. "Eli Lilly fined nearly $1.5B in drug marketing case," http://money.cnn.com/2009/01/15/news/companies/eli_lilly/.

[9] CNN. "Drug giant Pfizer to pay record $2.3B fine," http://www.cnn.com/2009/BUSINESS/09/02/Pfizer.fine/index.html.

[10] Information Systems Audit and Control Association. "Top Business/Technology Issues Survey Results," http://www.isaca.org/Template.cfm?Section=Home&template=/ContentManagement/ContentDisplay.cfm&ContentID=43978.

[11] FDA. "Regulatory Information," http://www.fda.gov/RegulatoryInformation/Legislation/default.htm.

[12] G. Cugola, and C. Ghezzi, "Software Processes: a Retrospective and a Path to the Future," *Software Process: Improvement and Practice,* vol. 4, no. 3, pp. 101-123, 1998.

[13] P. Abrahamsson, O. Salo, J. Ronkainen *et al.*, "Agile Software Development Methods," *Relatorio Tecnico, Finland*, 2002.

[14] R. N. Charette. "Why Software Fails," http://spectrum.ieee.org/computing/software/why-software-fails.

[15] N. Sridhar, M. RadhaKanta, and M. George, "Challenges of Migrating to Agile Methodologies," *Commun. ACM,* vol. 48, no. 5, pp. 72-78, 2005.

[16] B. Boehm, and R. Turner, "Observations on Balancing Discipline and Agility," *In Proc. of the Conference on Agile Development*, pp. 32–39, 2003.

[17] "Manifesto for Agile Software Development," http://agilemanifesto.org/.

[18] B. Kent, and A. Cynthia, *Extreme Programming Explained: Embrace Change (2nd Edition)*, p.^pp. 61-83, 111-117: Addison-Wesley Professional, 2004.

[19] K. Schwaber, "Scrum Development Process," OOPSLA Workshop on Business Object, 1997.

[20] J. De Luca, "A Practical Guide to Feature-Driven Development," Prentice-Hall, Englewood Cliffs, 2002.

[21] A. Cockburn, *Crystal Clear a Human-Powered Methodology for Small Teams*: Addison-Wesley Professional, 2004.

[22] C. Larman, *Agile and Iterative Development: A Manager's Guide*: Pearson Education, 2003.

[23] M. Fowler, and K. Beck, *Refactoring: Improving the Design of Existing Code*: Addison-Wesley Professional, 1999.

[24] Eclipse Process Framework "XP Model Library," http://www.eclipse.org/epf/downloads/xp/xp_downloads.php.

[25] P. Grünbacher, and C. Hofer, "Complementing XP with Requirements Negotiation." pp. 105-108.

[26] L. Crispin, and K. S. Rosenthal, *Testing Extreme Programming*, Pearson Education, 2002.

[27] F. McCaffery, D. McFall, P. Donnelly *et al.*, "A software process improvement lifecycle framework for the medical device industry." *In Proc. of the 12th IEEE International Conference and Workshops on Engineering of Computer-Based Systems,* pp. 273-280, 2005.

[28] F. Mc Caffery, J. Burton, and I. Richardson, *Software Process Improvement for Medical Industry*, 2005.

[29] G. Wright, "Achieving ISO 9001 Certification for an XP Company," *In Proc. of Extreme Programming and Agile Methods - XP/Agile Universe 2003*, pp. 43-50, 2003.
[30] ISO, "The International Standard for Quality Management," http://www.isoqar.com/iso9001/qualintro.htm.
[31] FDA, "Medical Device Use-Safety: Incorporating Human Factors Engineering into Risk Management," H. a. H. Services, ed., 2000.
[32] FDA, "Design Control Guidance for Medical Device Manufacturers," H. a. H. Services, ed., 1997.
[33] FDA, "Do It by Design: An Introduction to Human Factors in Medical Devices," H. a. H. Services, ed., 1996.
[34] F. Paetsch, A. Eberlein, and F. Maurer, "Requirements Engineering and Agile Software Development," *In Proc. of the 12th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2003.
[35] FDA, "Glossary of Computer Systems Software Development Terminology ", 2009.
[36] I. Sommerville, and P. Sawyer, *Requirements Engineering: A Good Practice Guide*, John Wiley Sons, Inc., 1997.
[37] I. Sommerville, *Software Engineering (7th Edition)*: Pearson Addison Wesley, 2004.
[38] A. Van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*, Wiley, 2009.
[39] K. Kowalczykiewicz, and D. Weiss, "Traceability: Taming uncontrolled change in software development," *Foundations of Computing and Decision Sciences,* vol. 27, no. 4, pp. 239-248, 2002.
[40] E. Hull, K. Jackson, and J. Dick, *Requirements Engineering*, Springer Verlag, 2005.
[41] R. Pressman, *Software Engineering: A Practitioner's Approach*, McGraw-Hill, Inc., 2003.
[42] U. Asklund, L. Bendix, and T. Ekman, "Software Configuration Management Practices for eXtreme Programming Teams," *In Proc. of the 11th Nordic Workshop on Programming and Software Development Tools and Techniques,* 2004.
[43] H. Andreas, "Usability engineering methods for software developers," *Commun. ACM,* vol. 48, no. 1, pp. 71-74, 2005.
[44] E. Folmer, and J. Bosch, "Architecting for usability: a survey," *The Journal of Systems & Software,* vol. 70, no. 1-2, pp. 61-78, 2004.
[45] E. Folmer, J. van Gurp, and J. Bosch, "Software Architecture Analysis of Usability," *Engineering Human Computer Interaction and Interactive Systems*, pp. 38-58, 2005.
[46] L. L. Constantine, and L. A. D. Lockwood. Process Agility and Software Usability: Toward Lightweight Usage-Centered Design, *Information Age, 8(8)*, August 2002.
[47] L. Jason Chong, and D. S. McCrickard, "Towards Extreme(ly) Usable Software: Exploring Tensions Between Usability and Agile Software Development," *In Proc. of the AGILE conference*, 2007.
[48] R. L. Nord, J. E. Tomayko, and R. Wojcik, "Integrating Software-Architecture-Centric Methods into Extreme Programming (XP)," *Carnegie-Mellon Univ Pittsburgh Pa Software Engineering, Inst*, 2004.
[49] D. Wells. "Extreme Programming," http://www.extremeprogramming.org/index.html.
[50] L. Cao, and B. Ramesh, "Agile Requirements Engineering Practices: An Empirical Study," *IEEE Software,* vol. 25, no. 1, pp. 60-67, 2008.
[51] A. Eberlein, and J. C. S. do Prado Leite, "Agile requirements definition: A view from requirements engineering," *In Proc. of the International Workshop on Time-Constrained Requirements Engineering,* 2002.
[52] A. Cockburn, and L. Williams, "The Costs and Benefits of Pair Programming," *Extreme programming examined*, pp. 223-248, 2001.
[53] P. Van Cauwenberghe, "Refactoring or Upfront Design?," *In Proc. of XP2001 Extreme*, 2001.
[54] E. M. Maximilien, and W. Laurie, "Assessing test-driven development at IBM," *In Proc. of the 25th International Conference on Software Engineering*, Portland, Oregon, 2003.
[55] T. C. Lethbridge, R. Laganiere, and C. King, *Object-Oriented Software Engineering: Practical Software Development Using UML and Java*: McGraw-Hill, 2002.
[56] N. Jaana, and K.-M. Mira, "Commonalities in Risk Management and Agile Process Models," *In Proc. of the International Conference on Software Engineering Advances,* 2007.
[57] L. Westfall, "Bidirectional Requirements Traceability," 2006.

[58] H. Obendorf, A. Schmolitzky, and M. Finck, "XPnUE--Defining and Teaching a Fusion of eXtreme Programming and Usability Engineering," *In Proc. of the HCI Educators Workshop on Teaching theory, design and innovation in HCI,* 2006.

[59] H. Obendorf, and M. Finck, "Scenario-Based Usability Engineering Techniques in Agile Development Processes," *In CHI '08 extended abstracts on Human factors in computing systems*, Florence, Italy, 2008.

[60] M. B. Rosson, and J. M. Carroll, *Usability engineering: scenario-based development of human-computer interaction*: Morgan Kaufmann Pub, 2002.