Investigating the Capability of Agile Processes to Support Life-Science Regulations: The Case of XP and FDA Regulations with a Focus on Human Factor Requirements

Hossein Mehrfard, Heidar Pirzadeh, Abdelwahab Hamou-Lhadj

Department of Electrical and Computer Engineering,
Concordia University, Montreal, QC, Canada,
{h_mehrfa, s_pirzad, abdelw}@ece.concordia.ca

Abstract Recently, there has been a noticeable increase of attention to regulatory compliance. As a result, more and more organizations are required to comply with the laws and regulations that apply to their industry sector. An important aspect of these regulations is directly related to the way by which software systems, used by regulated companies, are built, tested, and maintained. While some of these regulations require from these systems to support a very specific set of requirements, others, the focus of this paper, are concerned with the process by which the system has been built. The Food and Drug Administration (FDA) regulations, for example, impose stringent requirements on the process by which software systems used in medical devices are developed. One particular focus of the FDA regulations is on having a user-centered approach for building software for medical devices through the use of well-known concepts in the area of human factor engineering. In this paper, we discuss these requirements in detail and show how Extreme Programming, an agile process, lacks the necessary practices to support them. We also propose an extension to XP, that if adopted, we believe it will address this particular need of the FDA regulations for medical device software.

Keywords: Regulatory Compliance, FDA Regulations, Extreme Programming, Software Engineering

1 Introduction

For many regulated companies, regulatory compliance has become an important part of their business regardless of geography and industry sector. There are a number of factors behind the recent increase of attention to regulatory compliance including corporate scandals and the need for accountability, the reliance on Information Technology (IT) solutions and the necessity to protect and secure sensitive information [1]. As a result, more and more authoritative rules (i.e., regulations, laws, standards, and guidelines) are introduced every year putting further constraints on the way companies are operated, managed, controlled, and governed. Examples of these authoritative rules include Sarbanes-Oxley Act (SOX), the Health Insurance Portability and Accountability Act (HIPAA), the Food and Drug Administration (FDA) regulations, etc. Many of these authoritative rules have a direct impact on the way software systems, used by regulated companies, are developed and maintained. For example, a data records management tool, used by a health institution which is required to comply with HIPAA, must support many data security features such as a password-based protection mechanism, different levels of data access control, and frequent backups and data reliability techniques.

The overall objective of our research is to study ways to help software companies cope with the increasing customer demand for software systems that satisfy a large set of regulatory rules. These rules impact software development in many ways. Some of them manifest themselves as functional requirements that need to be supported by the final product. Many other regulations such as life-science regulations, and particularly FDA (the focus of this paper), are concerned with the process by which the software has been built. They define a set of artefacts, which vary significantly in coverage and depth that the process needs to produce in order for the resulting system to be compliant. Producing such artefacts would normally be feasible if one adopts a traditional process. However, traditional processes come with their own set of challenges such as a lack of flexibility to react to changing requirements. Agile software processes, which are popular alternatives, favour flexible development mechanisms but suffer from lack of documentation [2, 3].

The objective of this paper is three-fold:

- Discuss the human factors requirements that FDA regulations impose on software processes for building software for medical devices.
- Discuss how agile processes, in particular XP, lack the necessary mechanisms to satisfy the FDA requirements for human factors requirements.
- Propose an extension to XP that can be used in software projects that require FDA compliance.

Organization of the paper: We review XP in Section 2. In Section 3, we propose a generic framework for extending software development methodologies to support life science regulations. In Section 4, we describe the application of our framework to the FDA medical device regulations against XP with the focus on Human Factors Engineering (HFE), which is one of the most important focuses of the FDA regulations. In Section 5, we discuss related work. Finally, in Section 6, we conclude the paper and show future directions.

2 Extreme Programming (XP)

Many practitioners consider XP as the symbol of the agile methodology. 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 can yield a successful software process [4].

As illustrated in Figure 1, XP starts with an Exploration phase. In collaboration with the programmer, the customer writes stories about the system features that he expects to be available for the first release. Programmer leads the customer in this process by raising specific questions (e.g., “is the story testable?”) [5, 6]. The Planning phase prioritizes the collected stories based on their business values for the following small release. The required time and efforts for a release is estimated on a release plan in this phase [5].

At the beginning of developing a release, customer picks up stories based on the business values that he assigns to those stories. Then programmers break down those prioritized stories to a number of tasks and estimate the required time and efforts for each task. Based on this estimation and the structural stories, the story cards are reprioritized again to produce an iteration plan for whole iterations [6]. For the first iteration, it is important to choose the stories that mandate the system structure (architecture) for consequent iterations [5]. During each iteration, the specific set of selected stories are implemented by pair of programmers and tested by performing acceptance testing (functional testing). The iteration is not considered successfully implemented until it passes the acceptance test, which is normally written by the customer for verifying that the system functionalities satisfy the customer’s needs. Moreover, during the productionizing phase, a set of additional performance and quality tests are conducted for the current release. Then, the approved release is documented and deployed to the customer [5, 6]. After deploying the first release to the customer, in the maintenance phase, the project should keep the release running for the customer by enhancing it and fixing its existing bugs while producing new iterations simultaneously [5].

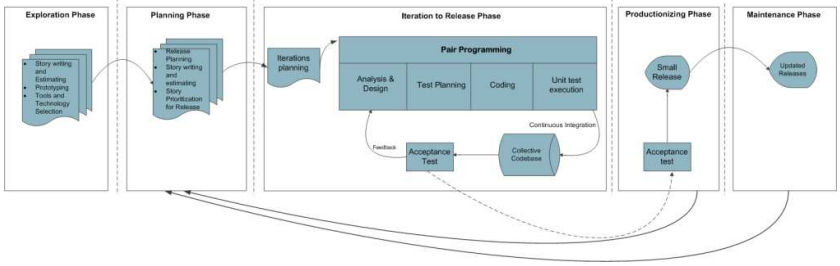

Fig. 1. XP Process Life Cycle (adapted from [5])

As Table 1 shows, XP consists of a number of roles, a set of practices, and work products. In the table, each role is composed of a number of sub-roles. For instance, the role "XP programmer" could be broke down to "XP architect", "XP interaction designer", "XP implementer", "XP programmer", and "XP integrator". However, this classification is not rigid and like many other software processes, it could be characterized based on the requirements of the project at hand. In our approach, we used the main references of XP [5, 6, 7] and the Eclipse Process Framework (EPF) model library for XP [8] to provide this table.

Table 1. The roles, practices and generated artefacts in XP

Role	Practice	Artefact
XP Coach	Adapt & improve process	None
	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	
	Refactor code	
	Write code	
	Help customer for story writing	
	Usage Evaluation	
	Large scale refactoring	
	System partitioning	
XP programmer(administrator)	Setup programmer environment	None
XP tester	Automate customer test	None
	Run customer test	
	Setup tester environment	
XP tracker	Track iteration progress	None
	Track release progress	

3 A Framework to Extend Software Methodologies to Support Life-Science Regulations

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

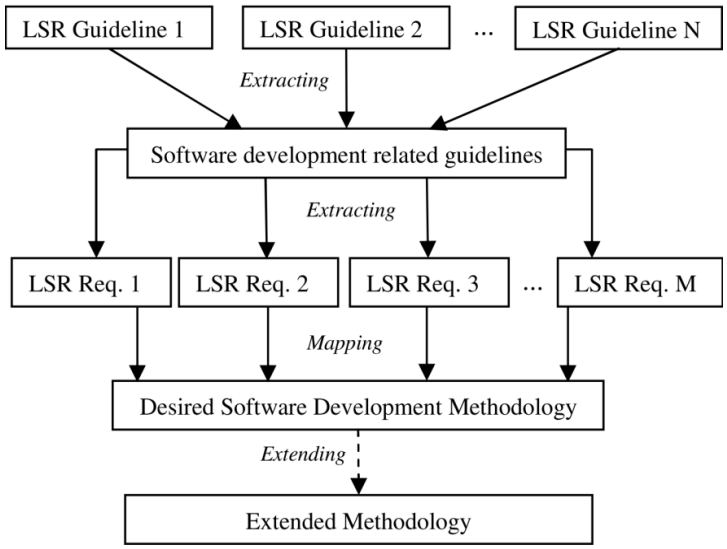

Fig. 2. A framework on how to meet life-science regulations for medical device software

LSRs often define the guidelines in a holistic way that is generic enough to be applied to various development methodologies. Unfortunately, this can cause lots of ambiguity for software developers as no specific development methodology can abide by the provided guidelines. For example, the FDA requests the medical de vice software developers to build safe and reliable software while no specific quality criteria are explicitly provided by the FDA.

Furthermore, LSRs often use 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 medical device guidelines as the developers do not know what they specifically have to follow.

In our approach, we alleviate the stated problems 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 2) is composed of the following steps:

1. We visit the LSR guidelines and extract the guidelines related to software development.
2. We study these requirements from the software process engineering perspective and present typical software practices and documentation that can help developers follow the LSR guidelines.
3. According to the suggested practices and documentation for the LSR compliance, we investigate the capabilities of our desired software development methodology for supporting the LSR requirements.
4. Based on our evaluation on how well our desired software development methodology can support the extracted requirements, we propose a possible extension of the methodology to support the missing requirements.

4 FDA and XP

The Food and Drug Administration (FDA) regulations, is a LSR that imposes stringent requirements on the process by which software systems used in medical devices are developed [9, 10, 11, 12, 13]. These requirements translate into various software artefacts that must be made available for the software to be FDA-compliant. In this paper, we discuss these requirements in detail and show its possible lack of capability of an agile process such as XP to meet these requirements. For this we took the steps mentioned in our generic framework. First, we went through the FDA guidelines associated with medical devices and extracted the guidelines related to software development. 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. For this, we took the following three steps to extract the software development requirements from the FDA:

1. We went through the guidelines for FDA medical devices and extracted the related software development requirements.
2. Among those extracted requirements, we collected software process related requirements.
3. We clarified each of FDA software process requirements by proposing a set of software practices and documentations for each of them. This requirements clarification is done by looking at the FDA requirements from a software engineering perspective.

Table 2 shows the results of taking these steps. As illustrated in this table, 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.

Table 2. FDA requirements for medical device software

Required Validation - Software Analysis -	**Methodology**	**Documentations**
Requirements Elicitation	Interview, use case, observation and social analysis, focus group, brainstorming, joint application development (JAD), 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, and risk analysis.	Not specified.
Requirements Traceability Analysis	Not specified.	Software requirements and system requirements traceability matrix, software requirements and the risk analysis result traceability matrix
Test Plan	Not specified.	Acceptance test plan, system test plan
Configuration Management	Version control	Not specified.
Required Validation - Software Design -	**Typical Practices**	**Typical Documentations**
Human Factors Engineering	Usability testing and inspection, architecture sensitive usability patterns, scenario-based assessment of architecture.	Not specified.
Software Design Evaluation	Prototype evaluation, demonstration, simulation, comparing the design with other similar 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	Not specified.	Requirement and design traceability matrix
Update Test Plan	Not specified.	Module test plan, integration test plan
Test Design Generation	Not specified.	Test case, test procedure
Configuration Management	Version control	
Update risk analysis	Risk analysis of software design using techniques such as medical device use description.	Not specified.
Required Validation - Software Coding -	**Typical Practices**	**Typical Documentations**
Source Code Evaluation	Code audit, code inspection, code walkthrough, code review	
Source Code Documentation Evaluation	Not specified.	Source code document
Code Traceability Analysis	Not specified.	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
Source code Interface Analysis	Interface checking	Not specified.
Test Generation	Not specified.	test case, test procedure
Required Validation Sub-Phases in Software Testing	**Typical Practices**	**Typical Documentations**
Test Documentation	Not specified.	Test plan, test procedure, test case, test report, test log.
Test Execution	Unit test, integration test, system test, user site testing	Not specified.
Test Traceability Analysis	Not specified.	Traceability matrices for: unit tests to detailed design, integration tests to high level design, and system tests to software requirements

Next, we investigated the capabilities of XP for supporting FDA requirements according to the suggested practices documentation for FDA compliance. For this, considering software process requirements that are extracted from FDA medical

devices' guidelines, we evaluated XP capability for supporting these requirements. The result of this evaluation is reflected in Table 3.

Table 3. Level of XP support for FDA requirements for medical device software

Development phases	Validation Sub-phases	XP Support for Typical Practices	XP Support for Typical Documentations
Requirements	Requirements Elicitation	User story card writing	No support for SRS
		High customer involvement	
		Eliciting requirements in number of iterations	
	Requirements Evaluation	System prototype	Not enough documentation
		Building software functionality in number of iterations	
		Handling most probable risks early in development	
	Requirements Traceability Analysis	No support	No documentation provided
	Test plan	Supported	Acceptance test plan
			No system test plan
	Configuration Management	No guideline	No documentation provided
Design	Human Factors Engineering	System prototype	No documentation provided
		Not enough support	
	Software Design Evaluation	Informal design review	No support for SDS and other documentations
		Not enough support	
	Design Traceability Analysis	No support	No documentation provided
	Update Test Plan	Supported	No documentation provided
	Test Design Generation	Supported	Unit test case
			Integration test case
			Acceptance test case
			No system test case
	Configuration Management	No guideline	No documentation provided
	Update risk analysis	No guideline	No documentation provided
Coding and Construction	Source Code Evaluation	Pair programming	No documentation provided
	Source Code Documentation Evaluation	No support	No documentation provided
	Code Traceability Analysis	No support	No documentation provided
	Source code Interface Analysis	No guideline	No documentation provided
	Test Generation	Supported	Unit test case
			Integration test case
			Acceptance test case
			No system test case
Testing	Test Documentation	Supported	Test Case
			No Support for the other documents
	Test Execution	Unit testing	No documentation provided
		Integration testing	
		Acceptance testing	
		System testing	
	Test Traceability Analysis	No support	No support

As illustrated in Table 3, XP lacks support for many FDA practices and requirements. Therefore, we extended XP for the missing FDA requirements. Before starting to extend XP to support FDA requirements we tried to identify possible challenges. Due to direct or indirect interactions of medical devices' software with human lives, FDA requires many practices as well as documentations for verification and validation of developing software. The importance of documentation for FDA derives from achieving high quality software for developers and the fact that FDA auditors need some level of documentation to approve the software. The FDA software process requirements are more fitted to the type of software processes called plan-driven processes such as RUP. Plan-driven process is a disciplined process for software development that relies on heavily documented knowledge and stringent practices [14]. On the other hand, XP emphasizes on less documentation and formality within the development life cycle. Thus, in extending XP we tried to reach a trade-off between keeping the process agile but at the same time inline with FDA. We extended XP by adding necessary sub-roles and practices that can support the

requirements that were missing. Although we studied in detail FDA requirements and whether they are supported by XP or not, in this paper, we only discuss our mapping process between FDA and XP by focusing on one important FDA requirement, which consists of the need for a process to support Human Factors Engineering practices – This is important for medical software since any error can cause human lives.

4.1 FDA and HFE

The FDA highlights the importance of Human Factors Engineering (HFE) during the software design process. The FDA defines human factors engineering as "a discipline that should be taken during software and hardware design to improve human performance in using medical equipments". This improvement should be in accordance with end users' abilities [12].

Considering the human factors engineering during the system design can result in a product which causes fewer design-originated human errors. 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 [12].

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.

In addition to HFE requirements for medical projects, FDA guidelines include HFE requirements during software development. The FDA highlights "software usability" as an HFE necessity to reach safety in software [9]. Furthermore, the FDA requires activities (e.g., usability test, risk analysis, prototyping and review) and produced documents (e.g., a test plan) during the software development process to accomplish the HFE requirement.

The FDA also provides a number of advices 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 example. 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 [12].

Next, this FDA general guideline needs to be rewritten from a software engineering point of view. For this, we need to have a concrete explanation of software usability during software design.

From the software engineering perspective, the usability of software is considered as a non-functional attribute that should be planed for during the development process [15]. Software Usability Engineering defines the usability of software based on seven subjective and objective characteristics: understandability, learnability, memorability, efficiency, low-error rate, compliance to standards and guidelines, and user satisfaction [16, 17]. These software characteristics are evaluated to measure the usability of the final software product.

Most of the existing usability techniques are suitable for the complete software system and do not measure usability in software architecture during development [17]. Based on usability definition, there should be techniques that are capable of assessing the usability of software during the design process. In addition, the usability of software is not limited to user interface design, rather depends on functionality of software such as undo functionality [17].

Moreover, there are sets of design solutions such as usability patterns and usability properties that increase usability of a software application, but these design solutions may cause changes to the software architecture [18]. To consider usability in software architecture design, numbers of architecture sensitive usability patterns are created that can be applied in high-level design such as actions on multiple objects, multiple views, and user profiles [18]. Moreover, there is a software architecture assessment technique called scenario-based assessment technique that provides early assessment of software architecture from usability point of view [18].

4.2 XP and HFE

The FDA requires high quality of usable software to reach HFE in software design. As mentioned before, the FDA suggests following HCI guidelines and usability engineering in software design. The XP is concerned about end users of software product by defining the role interaction designer. Interaction designer as the sub-role of XP programmer is responsible for evaluating usage of the deployed system. This evaluation results in to specify future functionalities of system by defining additional possible user stories. In addition, interaction designer refines the user interface according to usage evaluation which is developed during several iterations to release [7, 19].

During exploration phase, XP does not mention how to deal with usability in the architectural design. To design the software system architecture, XP suggests building system prototype in exploration phase to evaluate possible architectures

of software, create the high level design (architecture) of software in exploration and planning phases, and finally the architecture is consolidated in first release [5, 7]. There are two practices in XP that affect the design of architecture: system metaphor and simple design. System metaphors are shared story to describe how the system works and simple design makes easier to understand each design component [20]. But there is nothing in XP about following architectural patterns and assessing usability in architecture.

Constantine and Lockwood [21] believe that XP advocate a user-centered design because of the dependency of XP on customer feedback, setting goals based on what customers want, and getting iterative rapid prototype makes the development team design a system that the customer wishes, but not necessarily what he really needs. One of the drawbacks of this approach is that it is not possible to satisfy all stakeholders when the project has many stakeholders. This is due to the complexity of dealing with multiple stakeholder requests that might be conflicting.

As a result we conclude that XP does not satisfy FDA expectations to provide enough practices to back up usability in software design. Despite the existence of interaction designer, XP practices are not enough to handle usability in software projects that they need considerable amount of design due to scale of project.

4.3 Extending XP to Support HFE

As mentioned earlier, the most important aspect of HFE is usability. Thus, we propose an extension of XP abided by usability inside process. We base our efforts on providing major user stories on exploration phase. Based on the work done by Obendorf et al., they defined sub phases during the XP exploration phase to meet software usability in design [22].

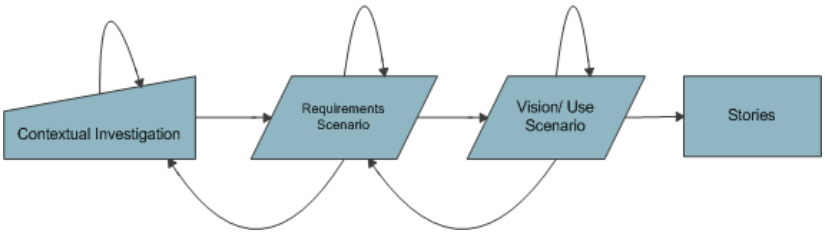

Fig. 3. The extended XP exploration phase

As illustrated in Figure 3, the exploration phase contains following sub-phases before prioritizing stories for each iteration: contextual investigation, requirements scenario, vision/use scenario and stories. Contextual investigation is the method for understanding application domain. This method comes from Contextual

Inquiry with less emphasis on completeness and modeling of gathered information. Contextual investigation uses interview and workplace observation to understand the use context, responsibilities and relationships of end users [23]. Requirement Scenario gets its basics from Problem Scenario in Scenario-Based-Engineering [24]. Requirement scenarios give details about the use of tools and their functionality which are already developed in specified workplace. The Vision or Use Scenario provides a consistent system look with problem statement and its corresponding solution for the specified system [23]. Additional investigation and feedback from users in each increment enhances the requirement scenarios and vision. The first three sub-phases result in to reach XP stories for that increment. In other words, the prioritization of stories for each increment is performed in the exploration phase of the extended XP.

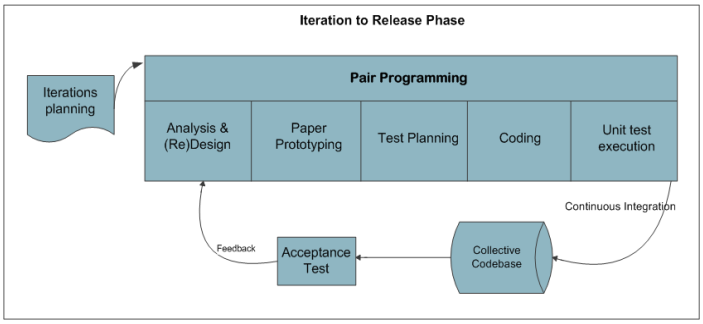

Fig. 4. The Extended XP Iterations to Release Phase

Besides extending the exploration phase in XP to handle usability in design, the extended XP could contain practices such as prototyping and redesigning in the Iterations to Release phase [23].

As mentioned before, in the first iteration during a release, the structural stories are chosen to consolidate architecture. Based on the achieved feedbacks by the end of each iteration, 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 to be able to redesign later. In addition, as we mentioned earlier in Section 4.1, using prototyping techniques during design increases the chance of developing usable software. Therefore, by doing paper prototyping after design in XP Iterations to release, the design model could be evaluated for its usability. The extended Iterations to Release phase is showed in Figure 4. We also suggest XP architect as the sub-role of programmer to become the main role who is responsible for applying usability disciplines for story card writing during exploration phase. XP architect can do so with high customer involvement. During exploration phase, this is the responsibility of customer to write use scenario and finally writing story card. In original XP, interaction designer is responsible for evaluating usage of

system during iterations to release phase. We suggest interaction designer (XP programmer sub-role) uses paper prototyping as a technique to facilitate usage evaluation of system. As a result of usage evaluation, he is redesigning the already designed features in next iteration according to this evaluation. This redesigning software does not need to change architecture. If XP interaction designer learned that existing architecture does not have the capability for redesign and architecture has to change, this is the responsibility of XP architect to change software architecture to state usability issues. Therefore, we suggest redesigning software features are being done under supervision of XP architect to not violate specified architecture.

5 Related Work

Kent Beck, one of the leading voices of XP, discussed the use of XP for developing secure and safe software in [7]. 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 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 concern of this paper.

In [25, 26], McCaffery et al. try to address the issue of compliance of medical devices with FDA regulations in process improvement level. For this they suggested the application of a software process improvement process like CMMI can ensure FDA regulatory compliance. Our work differs significantly from their in several respects: our generic framework is not limited to a specific Life-Science regulation. Furthermore, we address the same issue in the level of software development processes by practically extending XP as an important software development methodology.

In [27], Wright explained how he achieved ISO 9001 certification [28] using XP in a software development 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 this paper is that FDA requirements vary significantly from those of ISO 9001. The FDA is concerned with every single activity of a process.

6 Conclusion and Future Work

In this paper, we assessed the ability of XP to meet FDA regulations, which impose stringent requirements on the way software is built. These requirements are in the form of artefacts that a software process must produce for the software system to be FDA-compliance.

Although, we studied the complete set of FDA requirements, we chose in this paper to discuss our mapping process between FDA and XP by focusing on Human Factor Engineering requirements that must be met by any software process that claims to be FDA-compliant. We showed how XP does not support this aspect, and proposed an extension to it.

We intend to work on a larger version of this paper where we discuss every FDA requirement and if and how XP supports it or does not support it. For these activities that are not supported by XP, we intend to propose extension that will consist of adding new roles, practices and artefacts.

References

1. Paul N. Otto, and Annie I. Antón. Addressing legal requirements in requirements engineering, *In Proc. of the Requirements Engineering Conference*, pp. 5-14, 2007.
2. Gianpaolo Cugola, and Carlo Ghezzi. Software Processes: a Retrospective and a Path to the Future, *In Proc. of the Software Process Improvement and Practice Conference*, pp. 101-123, 1998.
3. Sridhar Nerur, RadhaKanta Mahapatra, and George Mangalaraj. Challenges of migrating to agile methodologies, *Communications of the ACM*, *48(5),* pp. 72-78, 2005.
4. Malik Qasaimeh, Hossein Mehrfard, and Abdelwahab Hamou-Lhadj. Comparing Agile Software Processes Based on the Software Development Project Requirements, *In Proc. of the International Conference on Innovation in Software Engineering*, pp. 49-54, 2008.
5. Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, and Juhani Warsta. *Agile Software Development Methods*, VTT Publications, 2002.
6. Craig Larman. *Agile and iterative development: A manager's guide*, Addison-Wesley, Boston, MA, 2003.
7. Kent Beck. *Extreme Programming Explained: Embrace Change*, Second Edition, Addison-Wesley, 2005.

8. Eclipse Process Framework, XP model Library, URL: http://www.eclipse.org/epf/downloads/xp/xp_downloads.php
9. FDA Guideline: General Principles of Software Validation; Final Guidance for Industry and FDA Staff, 2002, URL: http://www.fda.gov/medicaldevices/deviceregulationandguidance/guidancedocuments/ucm085281.htm
10. Glossary of Computer Systems Software Development Terminology, URL: http://www.fda.gov/ICECI/Inspections/InspectionGuides/ucm074875.htm
11. FDA Guideline: Design Control Guidance for Medical Device Manufacturers, 1997, URL: http://www.fda.gov/downloads/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/UCM070642.pdf
12. FDA Guideline: Do It by Design: An Introduction to Human Factors in Medical Devices,1996, URL: http://www.fda.gov/downloads/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/UCM095061.pdf
13. FDA Guideline: Medical Device Use-Safety: Incorporating Human Factors Engineering into Risk Management, 2000. URL: http://www.fda.gov/downloads/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/ucm094461.pdf
14. Barry Boehm, and Richard Turner. *Balancing Agility and Discipline: A Guide for the Perplexed*, Addison Wesley Professional, pp. 26-37, 2003.
15. Jakob Nielsen. *Usability Engineering*, Academic Press, Boston, 1993.
16. Andreas Holzinger. Usability Engineering Methods for Software Developers, *Communications of the ACM, 48(1),* pp. 71-74, 2005.
17. Elke Folmer, and Jan Bosch. Architecting for Usability; a Survey, *The Journal of Systems & Software, Elsevier, 70(1),*pp. 61-78, 2004.
18. Elke Folmer, Jilles van Gurp, and Jan Bosch. Software Architecture Analysis of Usability, *Lecture Notes in Computer Science, Springer,* pp. 38-58, 2005.
19. Jason Chong Lee, and Scott McCrickard. Towards Extreme(ly) Usable Software: Exploring Tensions between Usability and Agile Software Development, *In Proc. of Agile Conference,* pp. 59-71, 2007.
20. Robert L. Nord, James E. Tomayko, and Rob Wojcik. Integrating Software-Architecture-Centric Methods into Extreme Programming (XP), *Technical Note, CMU/SEI,* 2004.
21. Larry L. Constantine, and Lucy A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design.* Addison-Wesley, 1999.
22. Hartmut Obendorf, Axel Schmolitzky, and Matthias Finck. XPnUE – Defining and Teaching a Fusion of eXtreme Programming and Usability Engineering, *In Proc. of HCI Educators Workshop, inventively: Teaching theory, design and innovation in HCI,* 2006.
23. Hartmut Obendorf, and Matthias Finck. Scenario-Based Usability Engineering Techniques in Agile Development Processes, *In Proc. of the International Conference on Human Factors in Computing Systems*, pp. 2159-2166, 2008.

24. Mery Beth Rosson, and John M. Carroll. *Usability Engineering: Scenario-based Development of Human-Computer Interaction*, Morgan Kaufmann, 2001.
25. Fergal McCaffery, Donald McFall, Peter Donnelly, Frederick George Wilkie, and Roy Sterritt. 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.
26. Fergal McCaffery, Peter Donnelly, Donald McFall, and Frederick George Wilkie. Software Process Improvement for Medical Industry, *Studies in Health Technology and Informatics,* IOS Press, pp. 117-124, 2005.
27. Graham 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.
28. The International Standard for Quality Management, ISO, URL: http://www.isoqarinc.com/ISO-9001-Quality-Management-Standard.aspx