Software Quality Assurance in XP and Spiral - A Comparative Study

Sajid Ibrahim Hashmi and Jongmoon Baik
Information and Communication University,
109 Munji-ro, Yuseong-gu, Daejon,
South Korea
{hashmi,jbaik}@icu.ac.kr

Abstract

Agile processes have been introduced to avoid the problems most of software practitioners have run up against by using traditional software development methodologies. These are well known for their benefits like focus on quality, early business value delivery, higher morale of stakeholders, and the reduced cost/schedule. Also, they can support the earlier and quicker production of the code by dividing the product into small segments called iterations. However, there are on-going debates about their flexibility to accommodate changing requirements and whether the productivity and quality of the agile processes is satisfactory for the customers or not. Previously available studies have mostly focused on comparing XP(eXtreme Programming) with some other Agile methodologies, rather than comparing it with traditional plan-driven software development methodologies. In this Paper, we identify the XP phases and practices, how they ensure product quality, and map XP phases against the Spiral model phases to prove that XP has built-in QA(Quality Assurance) practices in its life cycle, in addition to its focus on productivity. A case study is also included to empirically investigate quality of the product developed using XP with comparison to the product developed using Spiral Model.

1. Introduction

Agile methods have been known to be effective software development processes which can result in high quality products. However, it is a controversial topic in software engineering. Many practitioners and researchers still doubt about its benefits, some are strongly against the agile development, while others suggest a mix of agile and plan driven development [1]. XP, which has been one of the widely used agile methods, concentrates on producing executable code and test drivers rather than focusing on software requirements and designs. This immense focus on source code makes XP controversial [2], but the fact is that XP is highly valuable in terms of simple design, emphasis on refactoring, testing, and code inspection by means of its related practices.

There are two of the main advantages of agile approaches: Firstly, they can effectively handle the unstable/changing requirements throughout the software development life cycle. Secondly, they can deliver the business values earlier by small increments of the product under budget and schedule constraints when compared to traditional plan-driven software development methods [3]. One of the most important issues is to satisfy customers with high quality products, which agile methodologies are supposed to resolve in an appropriate way. Is it possible to satisfy the customer with high quality products throughout the QAs of agile methods,? Is the level of QA in XP the same as that in Spiral? In order to address this issue, we compare the quality assurance techniques of both XP and Spiral development models.

A recurring aspect of many XP practices is a focus on product development risk control (or management). Instead of producing an artifact to satisfy the process, XP encourages developers to ask the question what would happen if the artifact is not produced. Apart from focusing on risks only, XP also focuses on building quality into the product rather than relying on a quality process that verifies a product after development. This built-in QA of XP is obtained by means of its practices like refactoring, metaphor, simple design, and Test Driven Development.

The primary purpose of this paper is two-fold. First, it identifies and describes the built-in QA practices in both XP and Spiral Model. Secondly, it provides a case study to investigate the relation of quality in both processes. Quality is chosen because this is one of the major issues for which XP is supposed to answer properly.

0-7695-2945-3/07 $25.00 © 2007 IEEE
DOI 10.1109/ICCSA.2007.65

Our research question is that quality is built into XP process, and XP addresses this issue better than any other traditional plan-driven software development method. The proposed approach consists of identifying the phases of XP, figuring out how those activities are involved at each phase, how they ensure the quality like Spiral model, and what is the empirical relation of quality in both processes. This approach can be helpful to investigate XP's support for software quality within its life cycle.

The rest of paper is organized as follows: Section 2 describes XP and Spiral processes in terms of stages and practices involved at each phase, Section 3 gives a life cycle comparison of both processes, Section 4 investigates the QA activities in Spiral and XP supported by a case study, and finally Sections 5 and 6 contain discussion and conclusion respectively.

2. Related Works

Currently available research works about XP varies in its nature in terms of the ways how quality is focused. Mainly practices like solo and pair programming have been investigated for productivity and quality. Mostly empirical data has been used by researchers to prove their hypotheses in favor of QA ability of XP; [2,14,15] have used different types of data to prove the quality support in XP; [20] has proved some drawbacks in XP based on some empirical data. [8] Has made life cycle comparisons between generic Agile and Waterfall process but there is not any empirical investigation to support the claim. On the whole, agile processes have not been investigated thoroughly against the plan driven methods. In general, empirical data has been used for the validations of results on QA of XP. However, the problem with these types of investigations is that data may vary from place to place and situation to situation. A same kind of validation may not hold true for all situations. Our approach is different in this sense. We have tried to make the validation based on the life cycle similarities of both XP and Spiral life cycles, rather than relying only on an empirical investigation.

2.1. eXtreme Programming (XP)

XP is one of widely used agile methods which can deal with unstable requirements by using a number of different techniques like simple planning, short iteration, earlier release, and feedback from customers on frequent basis. These characteristics enable agile methods like XP to deliver product releases in much shorter cycle time. XP was developed to address the needs of small teams who have been confronted with vague, unstable, and changing requirements. It has gotten four core values that are used to guide its employed practices. These values include communication, simplicity, feedback, and courage (or morale). A typical XP project has been applied to small projects with less than 20 developers, and it is mostly used for the projects without any strong base [2], i.e. without much documentation. It is important to understand the different phases of XP and role of each phase in order to establish the link with any other software process. Life cycle of an ideal XP project consists of the following phases [4].

- **Exploration:** It deals with ensuring that one is capable of going into production using XP. During the phase, programmers experiment with the limitations of the technologies they are supposed to use. They also experiment every programming task to figure out the exact time it would take while they are developing the product; meanwhile the customer is busy in writing the stories for the desired system.
- **Planning:** Here customers and programmers agree upon the date by when the smallest set of "stories" will be completed. Planning for the first release should be between 2 to 6 months; this commitment schedule is further broken into the iterations of one to four weeks duration. Each iteration will also produce the set of functional test cases for each of the stories.
- **Productionizing:** It tightens up the feedback cycle. Performance of system must also be tuned up for this phase. It is recommended to slow down the software production at this stage so that the risks become more evident for evaluation and mitigation.
- **Maintenance:** It deals with keeping the existing system running while at the same time producing the new functionality. The important thing about maintenance is that you have to be prepared for development interruption in case any production problems occur; but on the other hand one must be careful enough to change the design.
- **Death:** Dying well is also important for A good life. If customers do not come up with new stories, then it is time to wrap up the system. Customer's happiness is very important for this kind of death. Another reason for death might be the inability of system to accommodate further stories.

QA in XP is achieved by means of its different practices like testing, refactoring, system metaphor, pair programming, and Test Driven Development (TDD). These core practices complement each other.Testing is normally carried out by means of TDD which determines not only the design but also ensures that written code is defect free. Refactoring deals with

ensuring that code is always kept simple so that the probability of faults is minimized. System metaphor gives an understanding of system architecture. Hence the possibility of things going wrong is reduced if development is carried out based on the architecture. Pair programming allows developers to work together and share each other s' knowledge and identify their mistakes. All this ensures that developed product is either fault free, or has minimum number of faults.

2.2. Spiral Model

Spiral model for software development was developed with the basis of the various refinements of the waterfall model [5]. It can accommodate most previous models as a special cases and further provides guidance for the selection of previous models which best fits the given software situation. The spiral model has four quadrants as shown in Figure 1.The first phase begins with the identification of the objectives of the product, its functionality, the alternative means of implementing that specific portion of the product, and the constraints imposed on the implementation of those alternatives. The next step deals with evaluating the alternatives relative to the objectives and constraints. If user interface risks strongly dominate program development risks, the next step would be an evolutionary development which is a specification of the overall nature of the product, or a plan for the next level of prototyping which is more detailed one. This risk consideration can lead to a project implementing only a sub set of the steps in the model. On the other hand, if the previous prototypes have already resolved all the performance related risks, the third quadrant follows the basic waterfall approach to incorporate the further incremental development [5]. Planning for the next phase starts after the end of this incremental approach.

Quality is built into spiral model by means of activities involved at each phase, like risk analysis, prototype development, development plan, validation and verification, integration and acceptance testing. Each phase ensures that development is not moved to the next phase unless the previous phase is satisfied in terms of its activities. There is thorough analysis of requirements and risks even before the development starts, which guarantees that system contains only those requirements that are feasible and possible to implement. Further more, development phase of spiral performs step by step analysis of the product which ensures that no faults are escaped.

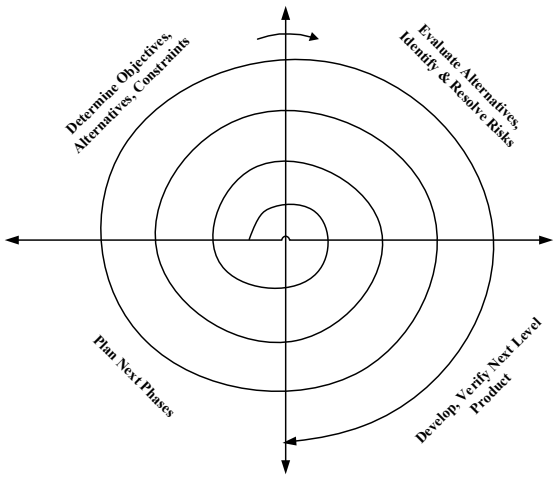

Figure 1. Spiral Software Process Model

3. Comparison of XP and Spiral Life Cycle

Spiral and XP are different models for software development, yet their practices in development sequence share some similarities. An agile process is a bit different from the traditional software development process in terms of life cycle practices because agile methods go through development stages little at a time because of short releases [4]. The steps may not be clearly separated as they are in traditional plan-driven software development methodologies. The Figure 2 illustrates a comparison between XP and spiral model in terms of activities involved.

Since both XP and Spiral are iterative in nature, it is possible to map phases of XP equivalent to phases of Spiral model. Phases of XP seem to have built-in similarities with the phases of Spiral model in terms of activities defined. The first phase of Spiral starts with Identification of objectives, alternative means of development, and constraints applied on such alternatives. The corresponding Exploration stage of XP deals with identification of constraints in terms of tools, techniques, and available resources. Continuous experimentation figures out the limitation and constraints of the approach and technology they are using. Next phase of Spiral model deals with evaluation of alternatives, identification and mitigation of risks. The stage 2 and 3 of XP serve almost the same purpose, especially in case of the risk identification and mitigation. Planning focuses on making agreements with customer for delivering the iterations. Test cases

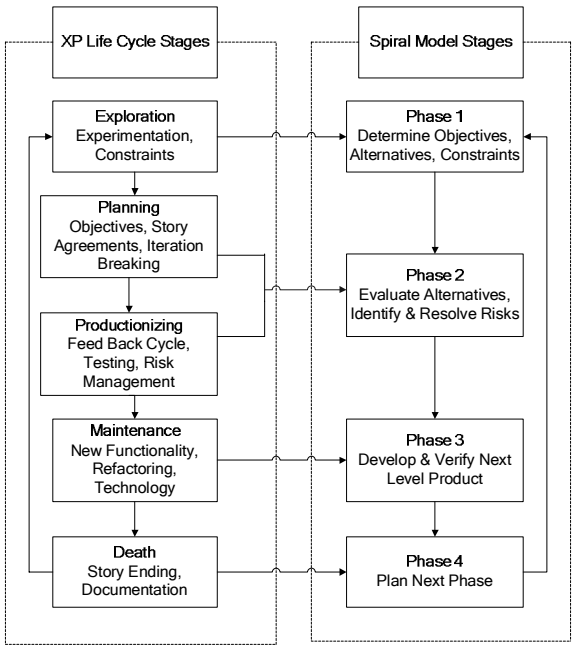

Figure 2. Spiral vs. XP Life Cycle

are produced for each iteration. Initially stories for those iterations are selected. Those stories represent the major part of the system; deviations from plans are monitored to check for consistency, and necessary actions are directed to be taken. Since a system prototype is developed, this stage is similar to the prototype development of the system in second quadrant of spiral model, where a system prototype is formed with important set of functionalities to check for anomalies during the system development. Furthermore, the developed prototype can be refined if desired results are to be achieved. Productionizing stage of XP carries on the identification of risks by focusing on feedback cycle. Software is allowed to go into production only when it is assured to be tested against risks. This stage of XP is slowed down so that risks become more prominent [4]. Next phase of the Spiral model concerns about the development and verification of the product. This is the phase where Spiral can accommodate any other model for software development. The main purpose of this accommodation of other process is to find a suitable solution for the software development. Corresponding Maintenance stage of XP ensures the production of new functionality while maintaining the existing code. Programmers use refactoring approach which basically involves formulating the solution for the programming problems [6]. Last phase of Spiral model integrates the things for planning the next cycle of spiral. The equivalent Death phase of XP is terminated when users' stories are finished and the cycle has to end. Like the last phase of Spiral, Death also produces some documentation which is related to the product.

4. Quality Assurance

What makes Spiral different from other approaches is its iterative approach and focus on risk mitigation. This approach may itself serve as a source of assuring quality, because major causes of failures and hindrances are identified and mitigated. The same case is true for the exploration phase in XP which focuses on the functionality of software, resources required, and plan for dealing with limitation of resources to achieve the desired functionality. After risks are mitigated, next phase of Spiral model can accommodate any kind of development model depending on the project characteristics; normally steps of Waterfall development model are being followed [5]. Waterfall model is an old model of software development and is still being used in industry. It includes well proven techniques such as peer reviews, code inspection, and unit and integration testing for quality assurance. It would make sense if we compare the remaining stages of the XP with waterfall method to prove the point that XP assures the product quality like Spiral model, even with short time periods. As far as Waterfall model is concerned, it uses both static and dynamic techniques, whereas XP mainly uses dynamic techniques [8]. XP can also adopt sub development methodologies, e.g. pair programming.

4.1. Quality Assurance in Spiral and XP

As mentioned earlier, Spiral model can accommodate other models (Waterfall) as a special case, it is feasible to compare it with XP. Waterfall model is termed as the simplest model for the software development. Its development activities include: system feasibility, software plans and requirements, product design with verification and validation, implementation with unit testing, integration and implementation with verification and system testing, and maintenance [7].

Figure 3 shows a Waterfall model with its supporting phases and activities. The inputs to the next phase should be validated by means of supporting quality assurance processes. Quality assurance is embedded into waterfall model at each phase by means of supporting practices. Once requirements are defined, they are validated and verified with the help of reviews, prototyping, and model validation.

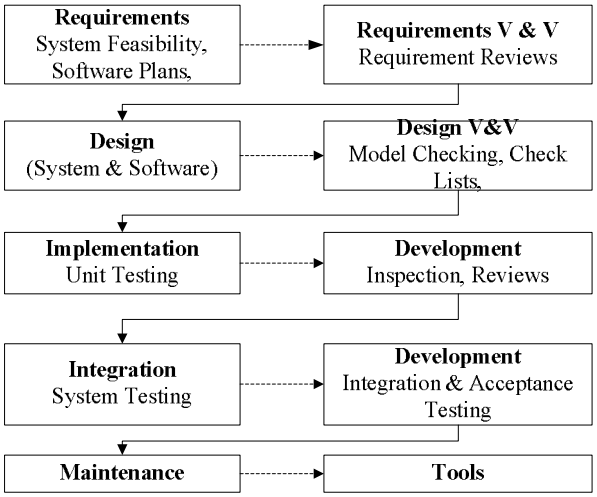

Fig 3. Waterfall Model with Supporting Practices

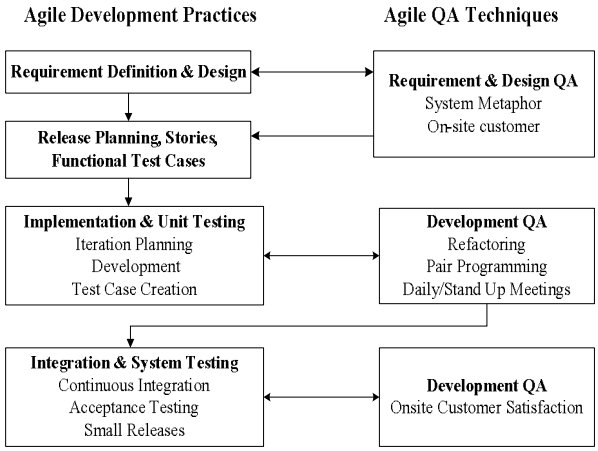

Figure 4. QA in Agile Methods

The generic development sequence of agile processes is the same as one in Waterfall model [8]. In XP, there are some practices that have both development and QA responsibilities. This means that developer may also have QA responsibilities. The development and QA practices collaborate with each other in order to maintain the pace of development. It means that QA practices are mingled up with development because of rush nature of XP; it is difficult to figure out the nature and the role these practices play in ensuring the product quality. By comparing the QA practices of XP with those of Waterfall model as a sub-process of Spiral model, it can be understood that quality measures are addressed in XP like Spiral model deals with them. For comparison, we take the QA practices of a generic agile method (Fig. 4). Any kind of agile development method focuses on the development of system metaphor which is the understanding of the whole System[9].It also emphasizes on customers understanding the metaphor along with the developers. The indirect benefit is that it increases the communication between customers and developers. In XP, communication is of utmost importance, and it is very difficult for an iteration process to result in project success if there is no communication. It also helps for the development of software architecture [10].

The requirement elicitation based on metaphor helps to elicit the correct requirements, also because of the fact that customer is present on the development site most of the times. It can help the developers throughout the development cycle. Customer involvement in XP is much higher than the requirement definition phase of Waterfall model; this increased involvement may help developers to straighten the path in case they are deviating from the actual requirements. Pair programming is a development methodology in which two programmers continuously work on the same code. It can also improve design quality and reduce defects. Developers under XP process also use refactoring for improving the design of existing software without changing its behaviors. Since each refactoring is small [8], the possibility of going wrong is also small.

Refactoring has got some advantages, which may serve the purpose of QA as well:

- It reduces the probability of errors in the system since developers are continuously restructuring the code which provides code inspection privilege.
- It minimizes the chances that a system gets faulty during the restructuring of refactored code.
- It requires continuous integration of code which catches faults in the software.
- It reduces the time developers have to spend on tracking the faults.

Acceptance testing is conducted as dynamic QA technique in XP. The difference between an XP and traditional acceptance testing is that occurring of acceptance testing is quite more frequent in XP as compared to traditional plan-driven software development methodologies. Earlier feedbacks from customer also serve as a precious characteristic of XP since it helps developers to get valuable information from customers. Development in XP is segmented in small iterations and these small releases are a good source of feedbacks; the shorter the release is, the quicker the feedback would be.

4.2. Empirical Results from a Case Study

The data used in this paper is obtained from different sources [11,12]. Four test iterations of spiral enhanced model are compared with XP project data to account for the level of QA obtained by both processes. The fault rates for both models are calculated as the sum of faults found during the analysis, test, and system integration.

4.2.1. Description of Case Projects

Table 1 provides the summary of two types of Case projects including their size, type, and iterations. Case 1 project used a modified form of Spiral model to develop different iterations of a product. The product in Case 2 followed an XP practice with main focus on pair programming. Developers did not have prior knowledge about the XP practice. The product in Case 1 was developed for external customer; Case 2 was developed for internal use. Case 1 involved professional developers and Case 2 involved both students and professionals. The size of the former was 66 KLOC, while the later had around 7.7 KLOC. Development language was "C" for the Case 1 and Java for the Case 2. The number of iterations was same for both cases. To convert the Case 2's data into KLOC/Year, we have assumed that there are 152 working hours in a month [13].

Table 1. Description of Case Projects

	Case 1	Case 2
Process	Spiral	XP
Product Type	Telecommunication System	Expert System
Size (KLOC)	66 KLOC	7.69 KLOC
Programming Environment	C	Eclipse/Java
Iterations	4	4

4.2.2. Comparison of Fault Rates

It is possible to compare the projects by driving the relative ratios for QA to prove the point that quality of XP project is not less than what is addressed by Spiral model.

Fault Rate: Table 2 shows the fault density per KLOC for each iteration of both Case projects. We have considered the fault rate for analysis, test, and system integration as a whole for each iteration of Case 1 so that total rate can be obtained.

Table 2. Comparison of Fault Rate

Fault Rate (Faults/KLOC)					
	1	**2**	**3**	**4**	**Total**
Spiral	5.84	5.44	8.41	7.97	6.91
XP	2.19	2.10	2.04	8.70	1.43*

*Some design faults are excluded

Figure 5 shows that fault ratios for both processes. It can be noticed that fault ratio for XP iterations is smaller than those for Spiral, except for the fourth iteration. The reasons for increased fault ratio in the fourth iteration of XP project is the less use of pair programming practice. The quality level of an XP project is also dependant on the extent to which its different practices are used.

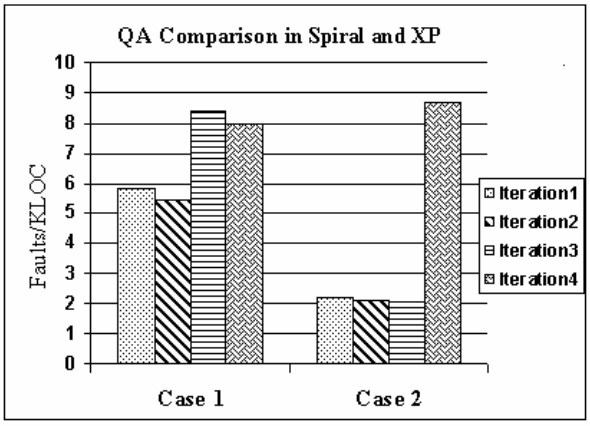

Figure 5. Spiral and XP (QA Comparison)

4.2.3. Validation

Student's *t*-test [14] is used to test the hypothesis we developed with our case study results. We use this test for comparing the means of two treatments, even if they have different number of replicates. In simple words, the *t*-test compares the actual difference between two means in relation to variation in the data. The *t*-test for quality is performed for both processes with the corresponding data as shown in Table 3.

Table 3. *t*-Test for Fault Ratio

	Case 1	Case 2
Iteration 1	5.84	2.19
Iteration 2	5.44	2.10
Iteration 3	8.41	2.04
Iteration 4	7.97	8.70
$\overline{X}$ Mean(total / n)	6.91	1.43*
σ^2 Variance	2.23	10.9

σ_d Standard deviation	2.25 $=\sqrt{\overline{\sigma_d^2}}$ (the standard deviation of the difference between the means).
$t = (\overline{X}_1 - \overline{X}_2 / \sigma_d)$	2.44 transpose $\overline{X}_1$ and $\overline{X}_2$ if $\overline{X}_2 > \overline{X}_1$ so that a positive value is obtained.

*some design faults are excluded

By entering the t-table [15] at 6 degrees of freedom ($n_1 + n_2 - 2$), we get 2.44 (normally p= 0.05) as the tabulated value of t, which is going up to a tabulated value of 5.96 *(p=0.*001). Tabulated value is somehow little greater (2.45) than our calculated value of t. It indicates that the difference between the two means is not highly significant. Clearly, the defect ratio of case 2 (XP) is less than case 1.

The analysis for *t*-test always considers variance, and it is valid only if variances of iterations/cycles are similar. There is a simple test to check if two variances are equal in statistical terms: divide the larger variance by the smaller (10.9/2.23=4.89) and compare the resultant variance ratio with a value from table of 'F' [16] for p=0.05. For two treatments, there is one degree of freedom between them. The tabulated F value is 6.0. Our variance ratio (4.89) is less than this. It means performed *t*-test was valid and both variances do not differ significantly.

5. Discussion

The results of empirical analysis showed that XP and Spiral address almost the same level of QA most of the times. As empirical data suggested, no significance superiority of one process to another process was detected. This is in contrast with existing studies that XP reduces the code quality because of changing requirements and less process artifacts. The examination of fault ratios of both development processes did not identify any significant difference between them. The findings of the case projects did not indicate that code produced by XP had more fault density in comparison with the second method. In fact, the inverse was true.

Quality Assurance is important to the quality of the software product regardless of the development process we choose. An XP cycle might not be as much detailed as the Spiral model cycle is; but considering the fact that an XP project consists of small iterations, an XP cycle can address the same concerns for an iteration what spiral model suggests for the whole project. XP consists of disciplined practices [18] testing, metaphor, refactoring, pair programming, continuous integration, that organizations can either formally introduce into their existing processes, or can use to supplement individual processes for project management, change management, requirement planning and testing.

Value based software engineering [19] has become highly attractive to practitioners in the recent years because of its emphasis on the value creation of the developed products. Another unique aspect of XP is the earlier delivery of business value to customers. In other words we can say that XP has closer link with value based software engineering, and key elements of VBSE [20] are addressed by XP. BRA (benefit realization analysis) is performed prior to starting iteration in the form of informal meetings where pros and cons of the project are discussed. These kinds of meetings are also helpful to identify the non software initiatives which may cause the realization of potential project benefits along with elicitation and reconciliation of stakeholders' value based conditions. The next element of VBSE deals with risk analysis and management that pervade the entire system life cycle. The productionizing stage of XP recommends slowing down the development so that risks can be identified and mitigated. Earned value management in VBSE tracks whether project is meeting its original plan. In XP, it is ensured in number of ways; e.g. system metaphor is established and refactoring is continuously performed to accommodate the changing requirements, without allowing them to affect the existing system behaviors. Above all, acceptance tests performed by on-site customer ensure that project never deviates from its proposed functionality, value measures are considered properly while product is being developed. XP has also got a capability to change as an opportunity, taking effect both from inside and outside. Changing requirements from customer can be the result of change in market trends, or introduction of new technology. Whereas XP itself is flexible enough to let its users use its core practices according to their requirements; e.g. pair programming can be introduced as a part of XP.

6. Conclusion

There are two main contributions of this research: it performs a detailed study to show that QA practices are built into an XP process like they are addressed by Spiral model. It provides empirical results related to Quality Assurance activities in XP. It can be concluded that an agile process like XP addresses quality issues repeatedly and continuously as compared to a traditional plan-driven software development method. Risk identification and mitigation is also carefully

addressed in each iteration. The reason is that in quality improvement activities like iteration planning, refactoring; feedback occurs frequently and continuously. These practices are not apparent in XP because there is very small gap among them, or we can say that these are on- going almost at the same time. Because of iterative nature of XP, the frequency of these activities is greater than Spiral model. XP focuses on continuous QA because of built-in QA activities repeatedly performed in each iteration and during the same iteration as well. The use of these QA activities is dynamic rather than static because an XP process goes into development phase quickly so it is not feasible to apply documentation based practices in it. Since development involves major activities like iterative development, refactoring, coding standard, collective ownership; developers are also responsible for performing QA activities which reinforces the quality of software in XP.

7. Future Work

In this paper, we provided an answer to the issue related to customer satisfaction in XP. Although there are some limitations of this research, i.e. like small sample size and no control over the case projects for the comparison. Strong conclusions cannot be based upon the results unless they are verified under different environments by considering different variables. The success or failure of any process lies in the success or failure of the end product delivered to the customer. Process management is an important task for any kind of process; it is also important for all the stakeholders to know the benefits of following a development process before they follow it. The link between QA practices and product quality must be to an extent that participants are motivated to follow the practices closely. Researchers are also encouraged to investigate the relationship between XP and Value Based Software Engineering.

8. Acknowledgements

This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Assessment).

9. References

[1]B. Boehm and R. Turner, "Using risk to balance agile and plan-driven methods," Computer, vol. 36, pp. 57-66, 2003.
[2]Frank Maurer, Grigori Melnik, "What You Always Wanted to Know about Agile Methods But Did Not Dare to Ask", ICSE '05, IEEE 2005.
[3]Barry Boehm, Richard Turner, "Balancing Agility and Discipline: A guide for Perplexed", Addison Wesley, August 2003.
[4]Kent Beck, "Extreme Programming Explained"; Addison Wesley 2000.
[5]Barry W. Boehm,"A Spiral Model for Software Development and Enhancement", Volume 21, Issue 5, May 1988 Page(s):61 – 72.
[6]James NewKirk, "Introduction to Agile Processes and Extreme Programming", ICSE '02, IEEE 2002.
[7]A. Abran and J. W. Moore, "Guide to the software engineering body of knowledge : trial version (version 0.95)." Los Alamitos, CA: IEEE Computer Society, 2001.
[8]Ming Huo, June Verner, Liming Zhu, Mohammad Ali Babar, "Software Quality and Agile Methods", COMPSAC '04, IEEE 2004.
[9]J. Grenning, "Launching extreme programming at a process-intensive company," Software, IEEE, vol. 18, pp. 27-33, 2001.
[10]Jean-Guy, Schneider, Rajesh Vasa, "Agile Practices in Software Development- Experiences from Student Projects", ASWEC '06, IEEE 2006.
[11]Noboru YAMAMICHI, Taka-aki OZEKI, Kouji YOKOCHI, Tetsuji TANAKA, "The Evaluation of New Software Developing Process based on a Spiral Modeling", 1996 IEEE.
[12]Pekka Abrahamsson, Juha Koskela, "Extreme Programming: A Survey of Empirical Data from a Controlled Case Study", ISESE' 04, IEEE.
[13]Barry W.Boehm, Ellis Horowitz, Ray Madachy, Donald Reifer, Bradford K. Clark, Bert Steece, A. Winsor Brown, Sunita Chulani, Chris Abts, "Software Cost Estimation with COCOMO II", Prentice Hall, 2000.
[14]http://helios.bto.ed.ac.uk/bto/statistics/tress4a.html.
[15]http://helios.bto.ed.ac.uk/bto/statistics/table1.html.
[16]http://helios.bto.ed.ac.uk/bto/statistics/table3.html.
[17]Barry Boehm, Richard Turner, "Management Challenges to Implementing Agile Processes in Traditional Development Organizations", IEEE Software, 2005.
[18]Arthur English, "Extreme programming, it's worth a look", ITProfessional, Volume 4, Issue 3, May-June 2002 Page(s):48–50,IEEE.
[19]Barry Boehm, "Value-Based Software Engineering: Overview and Agenda",USC-CSE-2005-504, February 2005, Copyright USC-CSE 2005.
[20]Barry Boehm, "Value-Based Software Engineering: Seven Key Elements and Ethical Considerations", USC-CSE-2005-503, February 2005 Copyright USC-CSE 2005.