An Empirical Evaluation of Iterative Maintenance Life Cycle Using XP

Jitender Choudhari
School of Computer Science & IT
Devi Ahilya University
Indore (M.P.), India
+91-9926090918
jeet_159@yahoo.co.in

Ugrasen Suman
School of Computer Science & IT
Devi Ahilya University
Indore (M.P.), India
+91-9826953187
ugrasen123@yahoo.com

ABSTRACT

Maintainability of a software product affects its maintenance cost and operational life. Maintainability of legacy systems, which have been developed through non-XP methodologies, has become a challenging issue for its maintenance. The iterative maintenance life cycle using extreme programming is an effective process for software maintenance [2]. This paper describes a controlled experiment that examines maintainability during maintenance of academic projects. The experiment was conducted with postgraduate students in a project course. The maintenance of each application was allocated to a couple of project teams; one team has used XP-based approach and yet another team has employed a traditional waterfall-based approach of maintenance. On measuring internal quality metrics of projects, it is observed that XP-based approach produces more maintainable code than traditional approach. The productivity of XP-based team is observed higher and at the same time, XP-based maintenance team was more confident about the code, and also reported higher confidence in future changes to their product. The iterative maintenance life cycle using XP has improved the maintainability of a software.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *Enhancements.*

General Terms

Standardization

Keywords

Software maintenance, iterative maintenance life cycle, extreme programming, maintainability

1. INTRODUCTION

Maintenance is an inevitable part in the life cycle of a software product. It consumes 40-90% of the total effort expended on a software system in its lifetime. Maintenance projects generally contend with unstructured code as a result of patched along with repatched while responding to successive client problems [1]. Modifications inside the piece of unstructured code without having proper check protection may raise hazards. Software maintenance process can be affected by staff turnover, lower crew spirits, poor visibility, complexity regarding repair projects, along with insufficient communication among stakeholders. The process of maintenance may be decrease through an absence of appropriate test suites [2, 3]. However, the literature implies that Extreme Programming (XP) methods used for maintenance projects provide effective solutions for aforesaid problems. TDD offers valor to help team members in the course of code-altering activities. At the same time, refactoring improves the extensibility and readability of the source code with the help of TDD. Other XP practices such as pair programming as well as collective ownership can offset the downsides that crop up from employee turnover. In addition, pair programming may make servicing chores more enjoyable, and it stimulates the pursuit of alternate options via continual assessment. Thus, maintenance professionals need a specific process model based on XP practices in the maintenance of non-XP projects. We have proposed an iterative maintenance life cycle using XP, a process model for software maintenance [2]. The cost of maintaining a software product and its operational life is highly influenced by its maintainability. Maintainability is an influential issue for the software developed through non-XP methodologies, which has legacy or unstructured code. Therefore, it is important to experimentally evaluate the effectiveness of iterative maintenance life cycle using XP on maintainability, productivity and other aforesaid issues of maintenance.

As maintainability of software is external quality attribute, there should be a direct method to measure maintainability. Software metrics may be used while indications intended for the measurement of external software quality characteristics [53]. Various empirical studies employ software solution metrics (internal quality attributes) while signs to determine external quality characteristics [6, 7, 8]. They suggested that most of object-oriented metrics can be useful quality indicators. Internal quality attributes consist of Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling between Objects (CBO), Response for a Class (RFC), Weighted Methods per Class (WMC), and Lack of Cohesion on Methods (LCOM) [9] and class size (Number of Methods and Lines of Code). A new link had been proven among external characteristics and internal quality metrics by Dandashi [10]. Also, Dandashi [10] confirmed a method regarding evaluating maintainability of object-oriented programs on the direct (internal) quality attributes including McCabe's Cyclomatic Number, RFC, CBO and many others. The internal quality attributes capture important dimensions of object-oriented design characteristics such as, inheritance by DIT and NOC; coupling computed using CBO, RFC, and FOUT; complexity calculated by WMC, NOM, and LOC; and cohesion measured by LCOM.

Confined experimental proof exhibits the effect of XP practices on maintainability during maintenance. This paper reports on a controlled experiment that was performed to examine the effects of the iterative maintenance life cycle using XP on maintainability and productivity during maintenance. The experiment was conducted with postgraduate students. Students in two groups worked on maintenance projects in a semester using either the iterative maintenance life cycle using XP or a traditional maintenance approach. In this experiment, we use internal quality metrics for measuring maintainability and comparing maintenance models. Teams were formed for both XP-based and traditional maintenance after the initial survey, attached in Appendix-A, related to team formation. After the experiment, a final survey was conducted that revealed the confidence in the code developed by each team and their experiences. This experiment revealed the differences between XP-based maintenance and traditional approach to maintenance with quality and productivity differences.

The process, phases, activities and artifacts in existing models of software maintenance are different from the iterative maintenance life cycle using XP. Therefore, a comparative analysis of the iterative maintenance life cycle using XP with traditional process models is required to understand the effectiveness of proposed model. The comparison is performed on the basis of various parameters such as, origin, requirement artifacts used, role of customer etc.

The rest of the paper is organized as follows; Section 2 discusses related work based on XP in an academic environment and an empirical study of development practices of XP from maintainability and productivity points of view. The description of experimental design is provided in Section 3. Experimental observations are presented in Section 4. Section 5 covers the empirical analysis. Discussion on the hypothesis is presented in Section 6. The effectiveness of the proposed model is discussed in Section 7. The concluding remarks and future work will be presented in Section 8.

2. RELATED WORK

Adaptation of XP in academics as well as the impact of XP practices on quality and productivity in product development or maintenance is important issues in empirical research. In the following sub-sections, literature on the adaptation of XP in academics for project course and empirical evaluation of XP practices from quality and productivity perspective are discussed and summarized.

2.1 XP in Academic Environment

Although XP is still quite new for academia, several studies have been carried out on the use of XP in educational environment that attempts to integrate XP into academic projects developed by the students to measure their applicability. Back et al. [11] have used XP as a development technique to carry out experiments in software engineering. They have suggested that a university setting can be the ideal place to perform practical experiments and test new ideas in software engineering, due to highly qualified research people and without the pressure of product release. They also report that pair programming has a great educational aspect such as learning from each other while working in pairs. Keefe and Dick [12] applied XP for academic projects and found that with the help of XP, students can gain useful skills throughout the project course especially less skilled students had shown more progress. They report that students fruitfully accomplished a system that met the client's requirements gathered throughout the planning game process.

Assassa et al. [13] performed an experiment and observed that XP teams produce the required product with full functionality and less effort. In addition, responses to change in requirements are more successful as compared to waterfall methodology. Following making use of agile approaches inside education of software engineering, Rico and Sayani [14] suggested that the success of agile requires prior introduction to the training of agile methods, teamwork, and development tools. Estellés et al. [15] select XP amid the entire methodologies of agile for ERP based final year project. They reported that the students have acquired knowledge with the customer and the company, which is impossible to acquire in the classroom. Dubinsky and Hazzan [16] make use of XP in degree course and present a structure for training of development methodologies. Noble et al. [17] reports their experience of incorporating an XP option into a project course. They reveal that the option of XP has been very successful with students producing more substantial software than the traditional approach, often with less work.

2.2 Empirical Evaluation of XP Practices

Several evaluative research studies have been conducted on XP (or individual practices) with professional practitioners in industry and by students in project courses to measure their effect on quality and productivity. This work falls into two main categories; namely, the adoption of XP practices in development, and attempts to adopt in the context of maintenance. These studies are summarized in Table 1, which includes important factors such as the environment, setting, effect on productivity, and effect on quality. Empirical evaluation of pair programming practice results higher productivity as compared to individuals. Evidences confirm that effort that's investing during pair programming will be non-linear with regards to developers' count [56].

Pair programming increases developers' problem solving abilities while working in pair. The abilities to solve problems save 40-50% of development time, thereby reducing time to quality assurance [18, 19, 20, 21]. Experimental outcomes show that the ratio of expenditure and productivity in pair is much better than solo developer [22, 23]. Pair programming improves team productivity during a new member addition by reducing the duration of fine-tuning and mentoring [24]. Jensen [25] claimed how the pair programming practice boosts productiveness by means of 127%. Another empirical study shows that pairs have a higher efficiency and overall productivity and it increases the business value of projects [26]. Pair programmers involve high effort due to pairs' incessant evaluation, improved ability of fault prevention, adherence to coding standards, along with the usage of refactoring. Studies show that pair programming adherence to coding standards provides shorter code, which improves readability and comprehensibility [28, 29, 30], thereby improving the software maintainability.

Empirical studies use software quality measures such as, simpler designs and defect density to evaluate impact of TDD in development and maintenance. TDD produces 18% better result for test cases in black-box testing. This reduces defect rate by 50% as compared to informal testing approach. TDD benefits into tasks together with minimum impact on efficiency in the course of advancement but additionally ends up with on time task completion [31, 32]. TDD method focuses on automated tests that will helps during maintenance and enhancements [33]. Studies also show that TDD provides better software quality by early defect reduction. The deficiency decrease is usually 45% much less each thousand line of code. Productivity is also increased due to significant test suites [34, 35, 36].

The cost as well as attempt involving software maintenance can be lessened by making use of refactoring. Refactoring provides activities that support restructuring by enhancing object-oriented design properties [38]. Various empirical studies demonstrate beneficial relativity among software quality and refactoring. Kataoka et al. [42] applied extract method and extract class activities of refactoring on a small C++ program and reported that the maintainability can be enhanced by refactoring. Their experiments used coupling measures to calculate maintainability. Another empirical study conducted by Moser [51] demonstrates enhancements of reusability using refactoring. Their experiments were conducted on a commercial system using Cyclomatic Complexity and CK measures.

Another empirical study as performed by Moser et al [52] on a commercial Java based software system that shows quality and an improvement of productivity. The experiment has used LOC, CK measures and effort (in hour). In addition they stated this refactoring diminishes code complexness and coupling with a boost regarding cohesion. Sahraoui et al [39] determined situations where refactoring can be used. They defined guidelines intended for the employment of coupling as well as inheritance steps associated with software quality to support ideal outcomes of refactoring. The impact of refactoring on program structure was analyzed by Bois and Mens [43, 44]. The study evaluated only three techniques of refactoring viz. Pull-up, extract method, and encapsulate field.The outcomes were being witnessed by using abstract syntax tree based framework on a modest tryout program.

Diverse scientific studies indicate that software extendibility could be enhanced by decreasing the size along with coupling measures reinforced by refactoring [40]. A study through Leitch and Stroulia [45] making use of extract and move function of refactoring tactics demonstrates improvement in software quality having decreasing size of code, dependancy occurrence and regression testing. Also, program performance can be improved by replacing conditional logics by polymorphism [41]. A quality framework proposed by [46, 47] uses design patterns to show high maintainability and better performing system. Many object-oriented measures were evaluated using the same framework [48]. The study and recommendations pertaining to bettering coupling along with cohesion methods and their particular purposes in refactoring optimizes program quality. Reduced change coupling using refactoring enhances software evolvability and do not lead to worse change coupling [49, 50]. Alshayeb [53] conducted a study to find the impact of refactoring activates on maintainability and demonstrates a relationship between them. Rech [54] proposed a quality assessment framework that addresses the problem of diagnosing and handling quality defects. The application of the framework is to recognize improvement of maintainability and productivity. Moser et al. [55] reported that XP enhances maintainability of software product.

From Table 1, it is concluded that most of studies evaluated XP practices individually for product quality and productivity evaluation as well as these are applied in development environment rather than maintenance. It is important to know the effect of XP practices on quality and productivity during maintenance.

We have conducted experiments on maintenance projects discussed in the subsequent sections that calculates maintainability and productivity. A comparative study is also presented between tradition and XP-based maintenance using code quality metrics.

Table 1. Empirical Study of XP Practices

Study	Environment	Practice	Setting	Effect on Productivity	Effect on Quality
Williams et al. [18]	Development	Pair Programming	Industry	40-50% higher	Improved software quality
Lui and Chan [20]	Development	Pair Programming	Industry	5% time savings	Provided better quality
Jensen [25]	Development	Pair Programming	Industry	Increased productivity by 127%	Produced a quality product
Williams et al. [26]	Development	Pair Programming	Industry	40-50% higher	Quality increased by 15%
George [31]	Development	TDD	Industry	Took 16% longer	Passed 18% more tests
Maximilien [32]	Development	TDD	Industry	Minimal impact	50% reduction in defect density
Williams [33]	Development	TDD	Industry	No change	40% reduction in defect density
Edwards [35]	Development	TDD	Academia	-	54% fewer defects
Kaufmann [34]	Development	TDD	Academia	Improved information flow	50% improvement
Erdogmus [36]	Development	TDD	Academia	Improve productivity	-
Sahraoui et al. [39]	Development	Refactoring	Industry	-	Improved maintainability
Stroulia and Kapoor [40]	Development	Refactoring	Academia	-	Increased extendibility
Demeyer [41]	Development	Refactoring	Industry	-	Improved program performance
Kataoka et al. [42]	Development	Refactoring	Academia	-	Enhanced system maintainability
Bois and Mens [43]	Development	Refactoring	Industry	-	Improved Internal Program Quality
Leitch and Stroulia [45]	Maintenance	Refactoring	Industry	-	Improved quality and reduce regression testing
Tahvildari et al. [47]	Development	Refactoring	Industry	-	Improved maintainability
Tahvildari and Kontogiannis [48]	Development	Refactoring	Industry	-	Improved maintainability
Bois et al. [49]	Maintenance	Refactoring	Industry	-	Improved quality
Ratzinger et al. [50]	Development	Refactoring	Industry	-	Enhanced software evolvability
Moser [51]	Development	Refactoring	close-to industrial	-	Improved reusability
Moser et al. [52]	Development	Refactoring	close-to industrial	Increases development productivity	Improved code maintainability
Alshayeb [53]	Development	Refactoring	Academia	-	Improved quality
Moser [55]	Development	XP	Industry	-	Improved maintainability

3. EXPERIMENTAL DESIGN

The goal of this experiment was to compare the iterative maintenance life cycle using XP with traditional maintenance approach for the purpose of evaluating maintainability and productivity. In this experiment, internal quality metrics was used for maintainability measurement. Mainly, three hypotheses were examined in this experiment. A formalization of the hypotheses is presented in Table 2. In this representation of hypothesis, maintainability of XP-based approach is labeled as $Maint_{XPM}$; whereas, maintainability of traditional approach is represented as $Maint_{TM}$. Productivity of XP-based approach is represented as $Prod_{XPM}$; whereas productivity of traditional approach is labeled as $Prod_{TM}$. Confidence about the code quality of XP-based approach is denoted as CCQ_{XPM}; whereas, confidence about the code quality of traditional approach is denoted as CCQ_{TM}. Hypothesis H1 tests if code produced by XP-based approach has higher maintainability than code produced by traditional approach.

Table 2. Formalized Hypotheses

Name	Null Hypothesis	Alternative Hypothesis
H1	$Maint_{XPM} = Maint_{TM}$	$Maint_{XPM} > Maint_{TM}$
H2	$Prod_{XPM} = Prod_{TM}$	$Prod_{XPM} > Prod_{TM}$
H3	$CCQ_{XPM} = CCQ_{TM}$	$CCQ_{XPM} > CCQ_{TM}$

Hypothesis H2 considers whether programmers of XP-based approach are more productive than programmers of traditional. Total maintenance time was considered for productivity evaluation. Hypothesis H3 tests programmers of XP-based approach have higher confidence about the code quality than programmers of traditional approach.

In this experiment, initially maintenance project is considered and an initial survey was performed for team selection. A training session on maintenance and XP was provided to the students. After the completion of maintenance activity, several metrics were collected and analyzed. The flow of experiment is shown in Fig.1 and discussed in subsequent paragraphs.

The experiment was conducted on five maintenance projects. These five projects described in Table 3 were developed by earlier batches of students by applying waterfall model of development. These applications are Exam Control-Room Management (ECM), Student Feedback System (SFS), Library Circulation System (LCS), Student Information System (SIS) and Campus Search Engine (CSE). After deployment phase, subsequent batches were involved in maintenance of these applications by applying either ad-hoc techniques or quick-fix approach. These applications were used to handle academic activity of various Departments in an Institute. The client needs advancements in the applications. Therefore, it is assigned the responsibility to new teams; each associated with a project coordinator and department in-charge with a target time line of six months.

The experiment was conducted with postgraduate students in a project course. In this experiment, maintenance tasks were performed by those PG students, who were not involved in the specification, design or development of the original systems. The maintenance of each one software had been allocated to a couple of project teams: one particular team used XP-based approach and yet another team employing a waterfall-based conventional procedure of maintenance. Both teams resolved exactly the same RC stories and worked underneath the similar circumstances. Implementation of RC stories had been carried out with incremental manner. Students had previous experience with the programming languages in which their respective applications were developed. They had never applied XP before neither do they have expertise in maintenance. Maintenance teams are prepared after the initial survey that ensures the teams were reasonably balanced. Initial survey is started through questionnaires that contain questions about knowledge of programming languages, level of experience in programming languages, awareness of XP practices and current and previous academic performance. After compilation of survey result, maintenance team selection is performed. A single maintenance project is assigned to two team, labeled as Team-1 (XP-based) and Team-2 (Traditional). Each team contains four to five members, therefore; the whole experiment involves forty to fifty students.

Table 3. Maintenance Projects Description

Project Name	Technology	Modules
ECM	Java, MySQL	Examination Superintendent
SFS	C#, MySQL	Student, Administrator
LCS	Perl, MySQL	Student, Librarian, Administrator
SIS	J2EE, MySQL	Student, Placement Department
CSE	J2EE, MySQL	Administrator, Faculty, Student

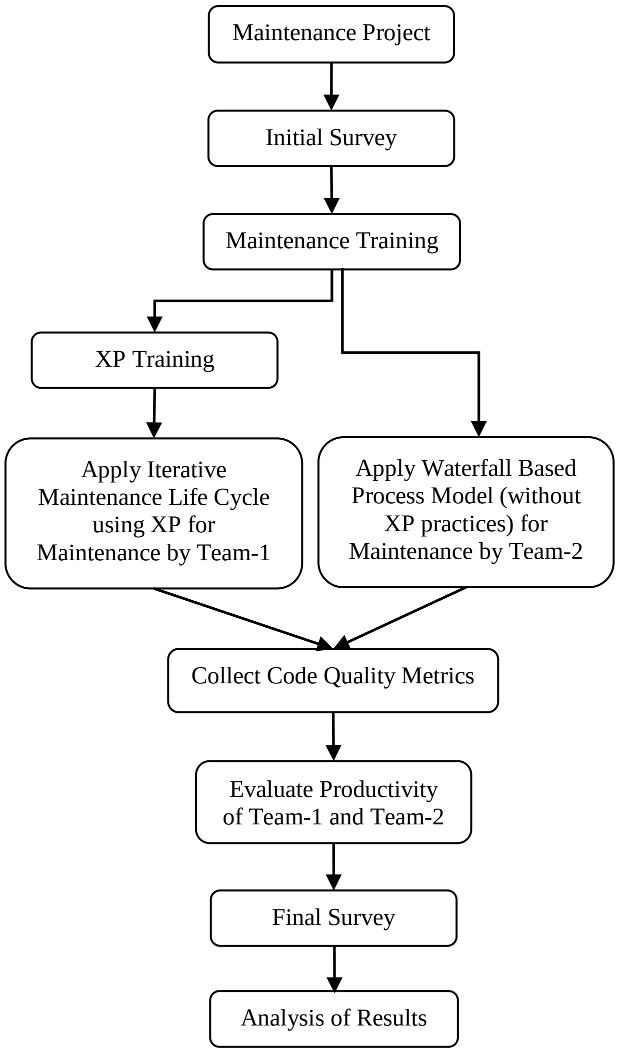
Maintenance Project
Initial Survey
Maintenance Training
XP Training
Apply Iterative Maintenance Life Cycle using XP for Maintenance by Team-1
Apply Waterfall Based Process Model (without XP practices) for Maintenance by Team-2
Collect Code Quality Metrics
Evaluate Productivity of Team-1 and Team-2
Final Survey
Analysis of Results

Figure 1. Flow of Experiment

After the team selection, a training session was provided to all maintenance team that contains concept and practice of software maintenance. Training material was also prepared and distributed to the students. After the completion of maintenance training, XP training was given to those teams that have to perform XP-based maintenance. Training contains introduction of agile and XP, concepts and practices of pair programming, refactoring and TDD in maintenance environment.

After the finish of experiments, number of several hours utilized in finishing RC stories had been accustomed to calculate the productivity of maintenance teams. As well as a final survey was conducted that revealed confidence in the code developed by teams. An analysis were performed which is based on values of code quality metrics and data of final survey.

4. EXPERIMENTAL OBSERVATIONS

After completing maintenance task, project coordinator and guides checked the working of systems as well as collected metrics using automated tool. Selected metrics as indicators for maintainability are calculated at the method, class, interface and project level after the experiment. Some metrics such as, McCabe Cyclomatic Complexity (VG), Nested Block Depth (NBD), Number of Parameters (PAR) are calculated at method and class level. Line of Code (LOC) is calculated only at method level. Number of Methods (NOM) is calculated at class and interface level. Lack of Cohesion of Methods_HS (LCOM) and Number of Children (NOC) are calculated at class and project level. Number of Static Methods (NSM), Specialization Index (SIX), Numberof Attributes (NOF), Method Lines of Code (MLOC), Weighted Methods per Class (WMC), Number of Overridden Methods (NORM), RFC (Response For a Class), Depth of Inheritance Tree (DIT) are calculated at class level. At the project level metrics such as Afferent Coupling (CA), Normalized Distance (RMD), Number of Classes (NC), Instability (RMI), Number of Packages (NOP), Number of Static Attributes (NSF), Abstractness (RMA), Number of Interfaces (NOI), Efferent Coupling (CE), CBO (Coupling Between Object classes) are calculated to measure the maintainability.

Production of maintenance teams can be determined by volume of several hours utilized in finishing RC stories. Confidence of maintainers in the code, which is modified by them, is checked by questions of final survey. The final survey was administered to all programmers. In the survey, maintainers were instructed to rank their own code through 1-5 scale.

Observations were collected for all maintenance projects, i.e., ECM, SFS, LCS, SIS, and CSE. For example, observations with respect to code quality metrics, which are analyzed during maintenance of ECM, are presented in Table 4 for both XP-based maintenance approach (XPM) and traditional maintenance approach (TM). In case of ECM, XPM produced code with low complexity in terms of VG, WMC, DIT, NOC, NBD and RFC as compared to TM and it is shown in Table 4. Value of VG is 2.41 in XPM and 4.10 in TM at method level. In the same way, the value of coupling metrics such as, CBO, CA, CE, and RMI are lower in XP-based approach. For example, value of CBO for XPM is 2.31 and value of TM is 4.42. As well as the value of cohesion metrics such as, LCOM and WMC indicate high cohesion in the system that is maintained using XP-based approach. These values of internal quality metrics indicate that during maintenance of ECM, XP-based maintenance approach produces more maintainable code than traditional approach of maintenance.

The team, which has performed maintenance of ECM using XPM, modified the systems in less time duration as compared to the team using TM as shown in Table 5. Completion time of maintenance activity for ECM project in XPM is 53 hours and in TM 85 hours. This significant difference shows that productivity of XPM team is higher than TM. Comparison between the confidence of programmers in code, which is drawn from final survey, is reported in Table 6. In ECM project, average value of maintainability for XPM team is 4.2 and value of TM is 3.1. This observation reveals that the team perceived the XP-based maintenance approach more confident about the code and also reported higher confidence in future changes to their product.

In case of SFS, XPM produced code with low complexity in terms of VG, WMC, DIT, NOC, NBD and RFC as compared to TM and it is shown in Table 4. The value of VG is 3.14 in XPM and 5.11 in TM at method level and value of VG is 3.67 in XPM and 4.16 in TM at the class level. The value of coupling metrics such as, CBO, CA, CE, and RMI are lower in XP-based approach. For example, value of CBO for XPM is 1.88 and value of TM is 2.12. Also, the value of cohesion metrics such as LCOM, and WMC indicate high cohesion in the system that is maintained using XP-based approach. These internal quality metrics indicate that the XP-based approach produces more maintainable code than traditional approach.

The XPM teams modified the systems in less time duration as it is shown in Table 5. Completion time of maintenance activity for SFS project in XPM is 68 hours and in TM 92 hours. This significant difference shows that productivity of XPM team is higher than TM. Comparison between the confidences of programmers in code is reported in Table 6. In SFS project, average value of maintainability for XPM team is 3.4 and value of TM is 2.8 and the average value of comprehensibility for XPM team is 4.1 and value of TM is 2.6. This observation reveals that the team perceived the XP-based maintenance approach more confident concerning the code.

5. EMPIRICAL ANALYSIS

Data of overall observations collected from experiments performed in previous section are summarized here to analyze the effectiveness of proposed XP-based process model. In this experiment, we have applied the proposed XP-based maintenance approach and traditional maintenance approach in maintenance of five different applications. Observations with respect to code quality metrics, maintenance effort, and programmers' confidence, which were analyzed during maintenance, are presented in Table 4 to Table 6. Here, statistics of observations are presented. As the maintainability of software product depends upon product attributes such as, complexity, coupling and cohesion; therefore, data under observation is grouped into these three categories. In this representation, complexity attribute is presented as the addition of metrics such as, VG, WMC, DIT, NOC, NBD, and RFC. In the same way, coupling attribute is the addition of metrics such as, CBO, CA, CE, and RMI. Cohesion is represented as an addition of LCOM and WMC metrics values.

Observations with respect to complexity metrics, which are considered during maintenance of all five projects, are presented in Fig. 2. In case of ECM, SFS and CSE projects, XP-based maintenance approach (XPM) produces less complex code in terms of VG, WMC, DIT, NOC, NBD and RFC as compared to traditional maintenance approach (TM) whereas, in case of LCS and SIS, TM produces less complex code. These values of complexity metrics indicate that the XP-based maintenance approach produces less complex code, which increases maintainability of code as compared to traditional approach of maintenance. Observation with respect to coupling metrics, which are considered during maintenance of all five projects, is presented in Fig.3, which are compared with XPM and TM. In all five projects, XPM produces low coupling code in terms of CBO, CA, CE, and RMI as compared to TM. These values of coupling metrics indicate that the XP-based maintenance approach produces low coupling code and hence, increases maintainability of code as compared to traditional approach of maintenance.

Table 4. Values of Code Quality Metrics

Metric	Level	ECM		SFS		LCS		SIS		CSE	
		XPM	TM	XPM	TM	XPM	TM	XPM	TM	XPM	TM
VG	Method	2.41	4.10	3.14	5.11	3.11	4.12	2.33	3.11	1.73	2.13
VG	Class	2.67	4.72	3.67	4.16	3.45	4.67	2.78	3.56	1.91	2.43
NSM	Class	0.0	0.06	0.04	0.08	0.0	0.0	0.0	0.06	0.06	0.08
CA	Project	5	12	7	11	4	5	6	5	4	8
RMD	Project	0.41	0.32	0.62	0.39	0.34	0.23	0.35	0.22	0.54	0.34
NC	Project	14	8	12	7	10	6	8	9	11	9
SIX	Class	0.81	0.43	1.21	0.34	0.58	0.18	0.67	0.25	0.88	0.42
RMI	Project	0.34	0.97	0.27	0.88	0.37	1.20	0.28	0.25	1.10	1.23
NOF	Class	5.2	4.3	3.5	3.9	6.2	4.1	4.12	3.11	6.88	4.13
NOP	Project	5	3	8	6	4	3	5	7	6	5
MLOC	Class	153.90	46.87	121.12	67.23	131.2	122.11	89.11	45.23	84.22	67.12
WMC	Class	4.8	10.4	3.60	8.12	4.12	6.14	5.13	4.21	3.91	5.23
NORM	Class	0.62	0.93	0.23	0.29	0.45	0.76	0.89	0.56	0.57	0.98
NSF	Project	0.0	4.0	0.41	2.63	0.81	3.12	0.0	0.92	0.42	0.89
NBD	Method	1.40	1.81	2.11	3.12	1.71	1.92	2.41	2.11	1.12	1.43
NBD	Class	1.21	1.85	2.21	3.16	1.41	1.78	1.56	2.11	2.14	2.24
NOM	Class	14	8	7	4	12	9	10	12	8	6
NOM	Interface	6	00	3	4	6	4	2	4	6	2
LCOM	Class	0.23	0.57	0.18	0.43	0.14	0.23	0.16	0.14	0.27	0.65
LCOM	Project	0.10	0.41	0.04	0.38	0.08	0.43	0.07	0.06	0.12	0.54
PAR	Method	0.64	1.80	1.34	1.78	0.89	1.2	0.56	1.85	1.23	1.98
PAR	Class	1.85	2.30	2.34	2.54	1.67	3.12	2.14	3.87	1.56	2.72
RMA	Project	0.26	0.13	0.53	0.34	0.67	0.24	1.24	0.45	1.67	0.62
NOI	Project	2	00	3	2	3	2	3	1	4	2
CE	Project	4.71	8.90	2.50	4.67	3.45	2.89	3.56	6.79	4.75	3.89
NOC	Class	1.67	1.23	2.13	1.78	1.30	1.10	1.87	0.76	1.65	0.92
NOC	Project	0.17	0.10	0.75	0.34	0.56	0.43	0.88	0.34	0.45	0.23
CBO	Project	2.31	4.42	1.88	2.12	3.13	2.15	1.87	1.91	2.45	4.87
RFC	Class	12.52	7.21	8.87	3.62	11.23	4.89	6.43	2.76	4.50	3.45
DIT	Class	1.12	1.23	1.45	1.24	2.13	3.67	2.13	3.43	1.43	1.78
LOC	Method	11	18	13	15	8	14	12	15	13	14

Table 5. Effort in hours

Parameter	ECM		SFS		LCS		SIS		CSE	
	XPM	TM	XPM	TM	XPM	TM	XPM	TM	XPM	TM
Maintenance Effort	53	85	68	92	86	104	35	47	42	65

Table 6. Programmers Confidence in the Modified Code

Parameter	ECM		SFS		LCS		SIS		CSE	
	XPM	TM	XPM	TM	XPM	TM	XPM	TM	XPM	TM
Maintainability	4.2	3.1	3.4	2.8	3.5	2.3	4.5	4.0	3.8	2.4
Comprehensibility	3.5	2.4	4.1	2.6	3.8	2.6	2.9	3.2	4.2	3.1
Reusability	3.2	3.2	3.5	3.1	2.9	2.5	3.8	2.4	3.1	3.6

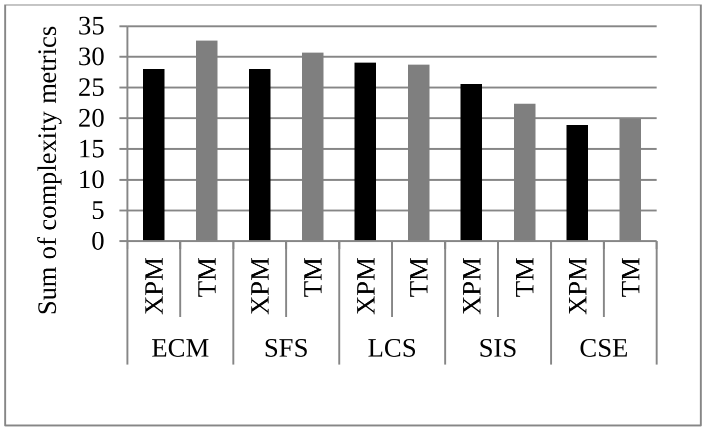

Figure 2. Complexity of Code

Observations with respect to cohesion metrics, which are considered during maintenance for all five projects, is presented in Fig. 4. It compares XPM and TM. In case of ECM, SFS, LCS and CSE projects, XPM produces lower values of LCOM and WMC as compared to TM whereas, in SIS project, TM produces lower values of LCOM and WMC as compared to XPM. These values of cohesion metrics indicate that the XP-based maintenance approach produces high cohesive code that increases maintainability of code than traditional approach of maintenance.

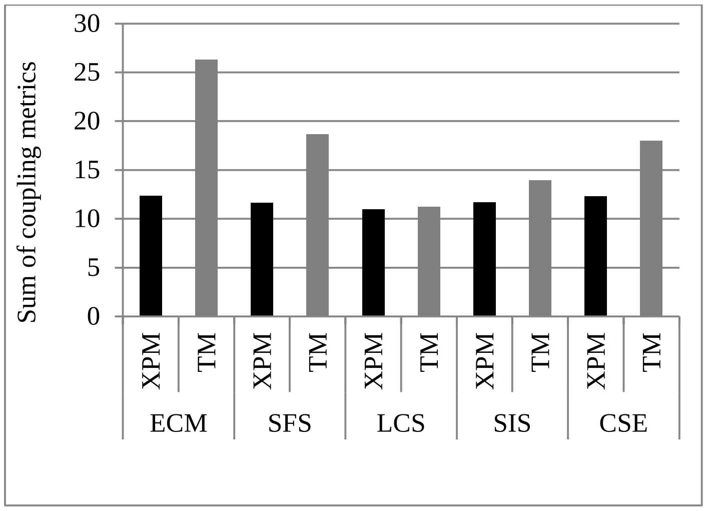

Figure 3. Coupling of Code

6. DISCUSSION

Maintenance task was performed for each of these academic projects using XP-based and traditional maintenance approaches. The values of code quality metrics, maintenance effort and programmers confidence were recorded as these are shown in Table 4 to Table 6. All the observations were recorded on the set hypothesis H1 to H3, which are shown in Table 2. We have observed the experimental results on the hypothesis H1 to H3, which are discussed in the subsequent paragraphs.

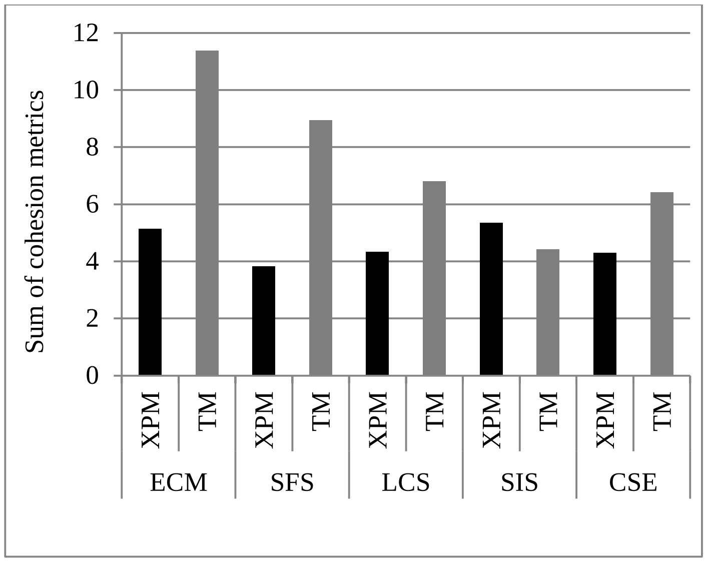

Figure 4. Cohesion of Code

Hypothesis H1: Is maintainability of the product is higher that it is maintained using XP-based approach?

After maintenance, more than thirty structural and object-oriented metrics were calculated to measure and compare maintainability of all five maintenance projects. The XP-based maintenance approach produced code with low complexity in terms of VG, WMC, DIT, NOC, NBD and RFC as compared to traditional maintenance approach. For example, in SFS project, the value of VG for XPM is 3.14 and value of TM is 5.11. In the same way, the value of coupling metrics such as, CBO, CA, CE, and RMI are low in XP-based approach. For example, in ECM project, the value of CBO for XPM is 2.31 and value of TM is 4.42. The value of cohesion metrics such as, LCOM and WMC indicate high cohesion in the system that is maintained using XP-based approach. These internal quality metrics indicate that the XP-based approach produces more maintainable code than traditional approach.

Hypothesis H2: Is productivity of programmer higher in XP-based maintenance approach?

It is observed that the XPM teams modified the systems in less time duration in all five maintenance projects. For example, completion time of maintenance activity for SFS project is 68 hours in XPM and 104 hours in TM. In the same way, duration of maintenance for CSE project in XPM and TM are 42 and 65 hours, respectively. This significant difference shows that the productivity of XPM team is higher than TM.

Hypothesis H3: Is programmer of XP-based approach has higher confidence about the code quality?

The final survey was administered to all programmers. Observations of this survey reveal that all five teams perceived the XP-based maintenance approach are more confident about the code and also reported higher confidence in future changes to their product. For example, average value of maintainability for XPM team is 3.4 and value of TM is 2.8 in SFS project.

7. COMPARISON OF MAINTENANCE PROCESS MODELS

The proposed process model is compared with existing process models of maintenance on the basis of parameters discussed in the subsequent subsection. The effectiveness of its different characteristics is discussed in subsequent paragraphs and it is shown in Table 7. The existing models of software maintenance are derived from the traditional models of software development. For example, origin of the quick-fix model is code-and-fix model and other ad-hoc development methodologies, Boehm model of maintenance is derived from spiral model of development, iterative-enhancement model of maintenance originated from iterative development models and full-reuse model of maintenance derived from the component-based development. IEEE-1219 and ISO-12207 maintenance processes are based on waterfall model and other traditional software development models. These maintenance process models have limitations in handling critical problems such as, unstructured code, team morale, poor visibility, and lack of communication between stakeholders. The proposed iterative maintenance life cycle using XP is derived from IEEE-1219 process of maintenance and extreme programming practices. This XP-based model resolves maintenance issues in a much improved manner. The proposed approach speeds up maintenance process and produces more maintainable code.

7.1 Parameters for Comparison

The maintenance process is applied in existing system and at the same time, it considers the organizational, technological, and other aspects for performing maintenance. The parameters used for comparing the iterative maintenance life cycle using XP with other maintenance models are discussed below.

Requirement artifacts: It shows the requirement artifacts used in problem reporting and change requirements elicitation.

Role of customer: Customer involvement is important aspects in XP as well as maintenance process. This parameter shows the involvement level of customers in the maintenance process.

Planning: Planning is an important activity of maintenance process. This parameter indicates the scope of planning such as, iteration planning containing limited change requirements, or a planning for entire set of requirements.

Number of iterations: It is the number of cycle considered to complete the list of change requirements.

Table 7. Comparative Analysis of the Iterative Maintenance Life cycle using XP with other Maintenance Models

Parameters	Existing Maintenance Models	Iterative Maintenance Life Cycle using XP
Requirement artifacts	Software problem reports, software change request form	RC story
Role of customer	Requirement submission and validation	Requirement submission, prioritization, validation
Planning	For whole set of change requirements	Iterative planning according to the business needs of customer
Number of iterations	Single or multiple	Multiple
Average duration of iterations	One to six months	Two-weeks
Estimation techniques	ACT model, FP model, COCOMO 2.0 reuse model	Planning poker, SMEEM
Design improvement practices	If required	Continuous process through refactoring, pair programming, and TDD
Programming style	individually programming	Pair programming
Code ownership	Module wise	Collective code ownership
Build time	At the end of module implementation, i.e., weekly or monthly	Continuous integration, i.e., daily basis
Code review practices	At the end of development	Continuous process through pair programming
Testing strategy applied	Unit testing, integration test and system test, regression testing, acceptance testing	TDD, continuous integration testing, acceptance testing
Execution of regression testing	After module completion	Frequent regression testing through continuous integration test
Amount of documentation	Huge	Limited
Phase execution	Sequential	Initial phase is sequential while remaining phase are parallel
Briefing style	Normal meeting	Daily stand-up meeting at story board
Application of process model	Any programming technology	Object-oriented programming technology

Average duration of iterations: It shows the average duration of iteration. It indicates the average time for delivery of working software integrated with the requested changes.

Estimation techniques: It describes the effort estimation techniques used at different stages of maintenance process.

Design improvement practices: It indicates the use of design improvement practices (for example, refactoring), and their rate of recurrence during maintenance.

Programming style: Maintenance is a tedious job and requires programming style to make it pleasurable. This parameter indicates agreeable style of programming (for example, pair programming).

Code ownership: It indicates whole or part experience and awareness of system among maintenance team. This experience may be of whole system or module-wise.

Build time: It is the time consumed for integration and can be considered as hourly, daily, weekly or monthly.

Code review practices: It indicates the use of code review techniques such as, peer-review, walkthrough, or inspection. These practices review the code for any errors or mistakes.

Testing strategy: It shows the types of testing approach used at different levels of maintenance and movement of testing, i.e., test-first or test-last.

Execution of regression testing: It reflects the way to verify the impact of modifications on other part of the same system.

Amount of documentation: It is the extent of standard documentation housekeeping during the maintenance process.

Phase execution: It shows the execution flow of phases in maintenance process viz. sequential, parallel, or hybrid.

Briefing style: It indicates the use of briefing style used during maintenance process.

Application of process model: It indicates the application area of process model; for example, system developed using object-oriented or procedure-oriented technology.

Problem reporting and its specification are important aspects of maintenance process and unambiguous change requirement and user acceptance criteria during maintenance reduces risks. Change requirements in existing software maintenance model are elicited and documented using existing artifacts, viz., software problem reports, software change request form etc. The proposed XP-based maintenance model uses RC story format for problem reporting during maintenance process. As RC stories are written by end users; hence, they are very helpful for elicitation of unambiguous change requirements. RC story enhances customer collaboration in addition to simplify the process of requirement elicitation process of maintenance. RC story eliminates the issues regarding weak awareness of the project and also deficit of communication throughout maintenance. RC story format contains acceptance criteria and recommended solutions. Acceptance criteria in RC story is helpful during acceptance testing to ensure that the components and the system as a whole provides expected results and recommended solution section supports for problem reproduction.

Involvement of customer is crucial in the success of development as well as during maintenance process. The existing process models involve customers during problem reporting and acceptance testing whereas, proposed XP-based maintenance model recommends their availability at the maintenance site for the problem reproduction, feedback, and acceptance testing. Customer submits RC story with the acceptance criteria and recommended solutions. Customer also takes the business decisions by participating in release planning with the developers. Involvement of customer improves understanding of system, and thereby improves the productivity of maintenance team.

Planning is an important activity in maintenance process. Existing models of maintenance plan for whole set of change requirements or a part of it. The proposed XP-based model of maintenance applies two levels of planning; namely, release planning and iteration planning. Release planning improves the involvement of customer to take business decisions. Iteration planning responds towards changes in requirements during maintenance.

Number of iterations used in maintenance process model to complete the list of change requirements is important to obtain the feedback that reduces risks. Existing models of maintenance have considered single iteration to complete all the change requirements that are submitted or can considered multiple iterations to complete it. The proposed XP-based model of maintenance completes all change requirements in multiple iterations. In XP-based model, initial iteration emphasizes on corrective maintenance and preventative maintenance in terms of code quality improvement. After initial iteration, perfective maintenance is performed according to the priority provided by the customer. Quality improvement habit of proposed model reduces future maintenance effort.

The duration of iterations indicate the average time for delivery of working software with the requested changes. The existing models of maintenance consumes single iteration to complete all change requirements submitted or can have multiple iterations to complete with average duration of maintenance being one to six months. The proposed XP-based model of maintenance completes all change requirements in multiple iterations with average duration of iteration being two week with reduced risks and provides rapid feedback.

Estimation methods and its application level throughout software process are extremely significant within the accomplishment regarding software projects. The existing models of software maintenance applies existing models such as, ACT model, FP model, COCOMO 2.0 reuse model etc., for the estimation of cost, size, and duration. These estimation models consider SLOC, FPs and object points as sizing units for determining maintenance effort. The proposed XP-based maintenance model recommends two level estimations. An initial estimation is performed during planning phase using planning poker technique. The second level of estimation is performed at the end of analysis phase using SMEEM. For accurate estimation, SMEEM model incorporates value adjustment factors for estimation of size and effort of a maintenance project. The algorithmic approach of SMEEM uses story points to calculate the volume of maintenance. SMEEM provides task and release level for estimation of different volume of maintenance. The task-level model of SMEEM estimates the effort of implementing each maintenance task, which is considered in the form of an RC story. This model deals with small effort estimates. The release-level model of SMEEM estimates the effort of a planned set of RC stories or a planned release. Through estimation using planning poker and SMEEM, the iterative maintenance life cycle using XP enfolds the nature of maintenance.

Approaches of coding in existing models of software maintenance provide very limited scope for design improvement practices. Literature shows that 5% of total cost is spent in preventative maintenance [144]. The proposed XP-based maintenance model provides scope for design improvement practices such as, refactoring, pair programming, and TDD. The code change approach that incorporates all these design improvement practices improves the maintainability of software product.

A change in the existing code, which is written by others is very common and lengthy job during maintenance of a software product. The existing models of maintenance support for solitary programming. The proposed XP-based maintenance model highly recommends pair programming. Pair programming induces pursuit of additional alternate options via continuous evaluate as well as tends to make maintenance tasks more pleasurable.

Staff turnover is one of the major problems of software maintenance. In the existing models of maintenance, maintainers work on a particular module without bothering about other modules of the same system. The proposed XP-based maintenance model highly recommends the collective code ownership and pair programming; and hence, maintainers gain the knowledge and experiences of whole system through team and partner switching. Problems, which are raised during maintenance due to staff turnover, are resolved by this type of working culture.

In the existing models of maintenance, integration is performed on weekly basis. The proposed XP-based maintenance model highly recommends continuous integration on daily-basis. This supports instant feedback and reduces testing effort after maintenance. Code review is an important aspect in maintenance of a software product. In existing models of maintenance, code review is performed after completion of a maintenance task. The proposed XP-based maintenance model supports continuous review through pair programming.

The suitable testing strategy plays a vital role in the success of a maintenance project. Existing models of maintenance apply different type of testing at different stages of maintenance such as, unit testing, integration testing, system testing, regression testing, and productive system testing. The proposed XP-based maintenance model applies different testing strategy. Test-first coding starts in change implementation phase. Change implementation phase also contains continuous integration and testing. Acceptance testing is applied on the basis of RC story acceptance criteria. Test-first approach delivers various advantages in maintenance process including, instant responses whilst modifying the legacy code, assurance along with valor while accomplishing error-prone changes, enhanced code legibility, and more rapidly impact analysis prior to any kind of changes. Continuous integration along with testing techniques associated with XP eliminates compatibility concerns and also interfacing issues associated with maintenance and provides a smoke testing environment to detect errors in early stages.

Regression testing is applied in existing models of maintenance to verify the impact of modifications. The proposed XP-based maintenance model does apply constant integration and testing to authenticate the result associated with changes. Continuous integration and testing curbs compatibility concerns and interfacing problems associated with maintenance supplying instantaneous suggestions to the maintenance team.

Existing models of maintenance uses huge amount of documentation during the maintenance process. For example, IEEE-1219 themselves suggests hefty documentation by means of 41 various IEEE specifications regarding software maintenance. The proposed XP-based maintenance model recommends very limited documentation during the maintenance. The proposed XP-based maintenance model emphasizes on code maintainability, satisfaction of customers and maintainers.

The proposed XP-based maintenance model recommends hybrid approach in which identification and planning phases are executed in sequential order while remaining phases can be executed in parallel according to the needs. Existing models of maintenance use sequential phase execution. Existing models of maintenance follow normal meeting for briefing whereas, the proposed XP-based maintenance model recommends daily stand-up meeting for briefing. This briefing style is used regarding the tasks for the day, technical issues and for information sharing among the team members. Existing models of maintenance can be used for application developed in any programming language. But the proposed XP-based maintenance model is suitable for those applications, which are developed using object-oriented technologies.

8. CONCLUSIONS AND FUTURE WORK

This study has evaluated the effect of XP-based maintenance on maintainability and programmers productivity along with the interest of programmers in maintenance and their confidence in producing code. It is observed that the XP-based maintenance approach may have a positive correlation with maintainability and programmers' productivity. The study also demonstrated that the programmers perceive XP-based maintenance approach more positive after exposing it, and particularly they are much more likely to adopt it in future. Final survey shows that programmers have higher confidence about the quality of code. The comparative analysis of XP-based model with traditional process models shows that XP-based model is the better approach than other models because it elicits maintenance needs using RC story that provides unambiguous change requirements with acceptance criteria. The RC story boosts client venture as well as resolves the down sides involving inadequate field of vision in the project, insufficient communication throughout the process of maintenance. The code change approach of XP-based model improves the productivity of maintenance team. Iterative planning of XP-based model improves the involvement of customer and responds to changes during maintenance. Two level estimation, i.e., planning poker at the initial level and SMEEM at the end of analysis phase, supports during maintenance planning. SMEEM offers much more realistic estimations. XP practices based code change approach improves the maintainability of software products through design improvement practices. Continuous integration and testing verify the impact of modifications that provides instant feedback. Apart from that, the proposed process model is an XP based technique; it offers documentation for upcoming maintenance. The iterative maintenance life cycle using XP consists of all features of a process model that can resolve problems of maintenance. Further study may be performed in larger populations and a broader base of programmers including students and practitioners that may provide the motivation for broader adoption of XP in maintenance environment.

9. REFERENCES

[1] Poole C. and Huisman, J. W. 2001 Using Extreme Programming in a Maintenance Environment. *IEEE Software.* 18, 2001, 42-50.

[2] Choudhari, J. and Suman, U. 2014. Extended Iterative Maintenance Life Cycle Using eXtreme Programming. ACM SIGSOFT Software Engineering Notes, Vol. 39, No. 1, January 2014, pp.1-12.

[3] Choudhari, J. and Suman, U. 2013. Code Change Approach for Maintenance using XP practices, The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, 131-136.

[4] Choudhari, J. and Suman, U. 2010. Iterative Maintenance Life Cycle Using eXtreme Programming. In Proceedings of the International Conference on Advances in Recent Technologies in Communication and Computing (Kottyam, India, October 15 - 16, 2010). ARTCom-2010. IEEE Computer Society, 401 – 403.

[5] Choudhari, J. and Suman, U. 2012. Designing RC Story for Software Maintenance and Evolution. In *Journal of Software (JSW),* Academy Publisher,7, 5, 1103-1108.

[6] Basili, V. R., Briand, L. C., and Melo, W. L. 1996. A validation of object-oriented design metrics as quality indicators. In *IEEE Transactions on Software Engineering*, 22, 10, 751-761.

[7] Li, W., and Henry, S. 1993. Object-oriented metrics that predict maintainability. In *Journal of systems and software*, 23, 2, 111-122.

[8] Gyimothy, T., Ferenc, R., and Siket, I. 2005. Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction. In *IEEE Transactions on Software Engineering,* 31, 897-910.

[9] Chidamber, S. and Kemerer, C. 1994. A metric suite for object-oriented design. In *IEEE Transactions on Software Engineering,* 25,5, 476–493.

[10] Dandashi, F. 2002. A Method for Assessing the Reusability of Object-Oriented Code Using a Validated Set of Automated Measurements. In *ACM Symposium on Applied Computing*, 997-1003.

[11] Back, R. J., Milovanov, L., Porres, I., and Preoteasa, V. 2002. XP as a framework for practical software engineering experiments. In *Proceedings of the Third International Conference on eXtreme Programming and Agile Processes in Software Engineering-XP*.

[12] Keefe, K., and Dick, M. 2004. Using Extreme Programming in a capstone project. In *Proceedings of the Sixth Australasian Conference on Computing Education,* Australian Computer Society, Inc., 30, 151-160.

[13] Assassa, G., Mathkour, H., and Al Dossari, H. 2006. Extreme programming: A case study in software engineering courses.

[14] Rico, D. F., and Sayani, H. H. 2009. Use of agile methods in software engineering education. In *Agile Conference, 2009. AGILE'09.* IEEE, 174-179.

[15] Estellés, E., Pardo, J., Sánchez, F., and Falcó, A. 2010. A Modified Agile Methodology for an ERP Academic Project Development.

[16] Dubinsky, Y., and Hazzan, O. 2005. A framework for teaching software development methods. *Computer Science Education*, 15, 4, 275-296.

[17] Noble, J., Marshall, S., Marshall, S., and Biddle, R. 2004. Less extreme programming. In *Proceedings of the Sixth Australasian Conference on Computing Education,* Australian Computer Society, Inc., 30, 217-226.

[18] Williams, L., Kessler, R. R., Cunningham, W., and Jeffries, R. 2000. Strengthening the Case for Pair Programming, *IEEE Software.* 17, 19-25.

[19] Williams, L. A. 2000. The Collaborative Software Process. PhD Dissertation, University of Utah.

[20] Lui, K. M. and Chan, K. C. C. 2003. When Does a Pair Outperform Two Individuals? *XP2003*, Italy.

[21] Müller, M. M. 2003. Are Reviews an Alternative to Pair Programming?. In *7th International Conference on Empirical Assessment in Software Engineering*, UK.

[22] Williams, L. 2001. Integrating Pair Programming into a Software Development Process. In *14th Conference on Software Engineering Education and Training*, USA.

[23] Ciolkowski, M. and Schlemmer, M. 2002. Experiences with a Case Study on Pair Programming, In *First International Workshop on Empirical Studies in Software Engineering*, Finland.

[24] Williams, L., Shukla, A., and Antón, A. I. 2004. An Initial Exploration of the Relationship Between Pair Programming and Brooks' Law, In *Agile Development Conference*.

[25] Jensen, R. W. 2003. A Pair Programming Experience, CrossTalk, In the *Journal of Defense Software Engineering*, 16, 22-24.

[26] Williams L. and Kessler, R. 2003. Pair Programming Illuminated: Addison-Wesley.

[27] Beck, K. 2006. *Extreme Programming Explained – Embrace Change*. Pearson Education Low price Edition Asia.

[28] Wood, W. A. and Kleb, W. L. 2003. Exploring XP for Scientific Research, *IEEE Software*, 20, 30-36.

[29] Gallis, H., Arisholm, E. and Dybå, T. 2003. An Initial Framework for Research on Pair Programming, *ISESE*, Italy.

[30] Cockburn, A. and Williams, L. 2000. The Costs and Benefits of Pair Programming. In *1st International Conference on Extreme Programming and Flexible Processes in Software Engineering*, Italy.

[31] George, B. and Williams, L. 2004. A structured experiment of test-driven development. *Information and Software Technology*. 46, 5, 337-342.

[32] Maximilien, E. M. and Williams, L.2003. Assessing test-driven development at IBM. In *Proceedings of the 25th International Conference on Software Engineering* (ICSE-03), Piscataway, NJ, IEEE Computer Society, 564-569.

[33] Williams, L., Maximilien, E. and Vouk, M. 2003. Test-driven development as a defect-reduction practice. In *Proceedings of the 14th IEEE International Symposium on Software Reliability Engineering*, 34-45.

[34] Kaufmann, R. and Janzen, D. 2003. Implications of test-driven development: a pilot study. In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, ACM Press, 298-299.

[35] Edwards, S. 2003. Using test-driven development in the classroom: providing students with automatic, concrete feedback on performance. In *Proceedings of the International Conference on Education and Information Systems: Technologies and Applications* (EISTA'03).

[36] Erdogmus, H. 2005. On the effectiveness of test-first approach to programming. In *IEEE Transactions on Software Engineering.* 31, 1, 1-12.

[37] Fowler, M. , Beck, K., Brant, J. , Opdyke, W. , and Roberts, D. 1999. Refactoring: Improving the Design of Existing Code, Addison Wesley.

[38] Shatnawi, R. 2010. A quantitative Investigation Of The Acceptable Risk Levels of Object-Oriented Metrics in Open-Source Systems. In *IEEE Transactions on Software Engineering.* 36, 2, 216-225.

[39] Sahraoui, H.A., Godin, R., and Miceli, T. 2000. Can Metrics Help To Bridge The Gap Between The Improvement of OO Design Quality And its Automation? In *International Conference on Software Maintenance*, 154-162.

[40] Stroulia, E., and Kapoor, R.V. 2001. Metrics of Refactoring-Based Development: an Experience Report. In *The seventh International Conference on Object-Oriented Information Systems*, 113-122.

[41] Demeyer, S. 2002. Maintainability versus performance: What's the effect of introducing polymorphism?. technical report, Lab. on Reeng., Universiteit Antwerpen, Belgium.

[42] Kataoka, Y., Imai, T., Andou, H., & Fukaya, T. 2002. A quantitative evaluation of maintainability enhancement by refactoring. In *International Conference on Software Maintenance,* IEEE, 576-585.

[43] Du Bois, B., and Mens, T. 2003. Describing the impact of refactoring on internal program quality. In *International Workshop on Evolution of Large-scale Industrial Software Applications,* 37-48.

[44] Mens, T., Demeyer, S., and Janssens, D. 2002. Formalising behaviour preserving program transformations. In *Graph Transformation,* Springer Berlin Heidelberg, 286-301.

[45] Leitch, R., and Stroulia, E. 2003. Assessing the maintainability benefits of design restructuring using dependency analysis. In *Proceedings of Ninth International Software Metrics Symposium,* IEEE., 309-322.

[46] Tahvildari, L. 2003. Quality-Driven Object-Oriented Re-engineering Framework. PhD Thesis. Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada.

[47] Tahvildar, L., and Kontogiannis, K. 2004. Improving design quality using meta-pattern transformations: a metric-based approach. In *Journal of Software Maintenance and Evolution: Research and Practice. 16,*4,5, 331-361.

[48] Tahvildari, L., Kontogiannis, K., and Mylopoulos, J. 2003. Quality-driven software re-engineering. In *Journal of Systems and Software*, 66,3, 225-239.

[49] Du Bois, B., Demeyer, S., and Verelst, J. 2004. Refactoring-improving coupling and cohesion of existing code. In *Proceedings of 11th Working Conference on Reverse Engineering,* IEEE, 144-151.

[50] Ratzinger, J., Fischer, M., and Gall, H. 2005. Improving evolvability through refactoring. In *ACM SIGSOFT Software Engineering Notes,* 30, 4, 1-5.

[51] Moser, R., Sillitti, A., Abrahamsson, P., and Succi, G. 2006. Does refactoring improve reusability?. In *Reuse of Off-the-Shelf Components,* Springer Berlin Heidelberg, 287-297.

[52] Moser, R., Abrahamsson, P., Pedrycz, W., Sillitti, A., and Succi, G. 2008. A case study on the impact of refactoring on quality and productivity in an agile team. In *Balancing Agility and Formalism in Software Engineering,* Springer Berlin Heidelberg, 252-266.

[53] Alshayeb, M. 2009. Empirical investigation of refactoring effect on software quality. *Information and software technology*, 51, 9, 1319-1326.

[54] Rech, J. 2009. Context-sensitive Diagnosis of Quality Defects in Object-Oriented Software Systems, Ph. D. Thesis. Hildesheim: University of Hildesheim, Department IV.

[55] Moser, R., Scotto, M., Sillitti, A., and Succi, G. 2007. Does XP deliver quality and maintainable code?. In *Agile Processes in Software Engineering and Extreme Programming,* Springer Berlin Heidelberg, 105-114.

[56] Hulkko, H. and Abrahamsson, P. 2005. A multiple case study on the impact of pair programming on product quality. In *Proceedings of the 27th international conference on Software engineering,* ACM, 495-504.

[57] Choudhari, J. and Suman, U. 2012. Story Points Based Effort Estimation Model for Software Maintenance. In *Proceedings of the 2nd International Conference on Computer, Communication, Control and Information Technology* (Hooghly, India, February 25 - 26, 2012). C3IT- 2012. Procedia Technology, 4, 761-765.

[58] Choudhari, J. and Suman, U. 2012. Phase wise Effort Estimation for Software Maintenance: An Extended SMEEM Model. In *Proceedings of the CUBE International Information Technology Conference* (Pune, Maharashtra, India, September 3 - 5, 2012). ACM., 397-402.

APPENDIX-A

Initial Survey

In the controlled experiment, maintenance teams are formed for both XP-based and traditional maintenance approach on the basis of this initial survey that ensures the teams were reasonably balanced. Tick mark in the choices or fill values in specified range, as the case may be.

What is your enrollment number in the university?

Specify your gender?

- □ Male
- □ Female

How old are you as on date?

- □ 18 to 22 years
- □ 23 to 26 years
- □ 27 to 35 years
- □ over 35 years

Name of the degree completed at graduation level?

- □ BCA
- □ B.Sc.(CS/IT)
- □ BE (CS/IT)
- □ Other, please specify _____

What is your marks in % at graduation level?

____________%

What is your Cumulative Grade Pointer Average (CGPA) in current course?

- □ 9.0 – 10.0
- □ 8.0 – 9.0
- □ 7.0 – 8.0
- □ 6.0 – 7.0
- □ 5.0 – 6.0
- □ 4.0 – 5.0

How many programming languages have you read at graduation level?

- □ 0
- □ 1
- □ 2
- □ 3
- □ 4 or more

How many programming languages have you read at school level?

- □ 0
- □ 1
- □ 2
- □ 3
- □ 4 or more

Specify your own level of proficiency with the following programming languages?

Programming Language	Tools Used	No. of Programs Write	No. of Projects Developed	Last Program Written Before	Overall Rating (1-5)
C					
C++					
Java					
J2EE					
.Net					
PHP					
JavaScript					
Any other (please specify below)					

Specify your own level of proficiency with the following XP practices?

XP Practice	Tools Used	Last Use Before	Overall Rating (1-5)
Planning game			
Small releases			
Metaphor			
Simple design			
Refactoring			
Pair programming			
Collective ownership			
40-hour week			
On-site customers			
Coding standards			
Test driven development			
Continuous integration			

If you had a choice of writing/ modifying code with traditional or XP practices based approach, which would you choose?

- ☐ Traditional
- ☐ XP practices based approach

Why above approach is selected?

Additional Comments:

Thanks for providing valuable information.

Final Survey

This final survey is conducted after completion of experiment that reveals confidence in the code developed by teams and their experience. Tick mark in the choices or fill values in specified range, as the case may be.

What is your enrollment number in the university?

Title of the maintenance project:

Number of members in your project team:

Which maintenance approach is used?

- ☐ Traditional maintenance
- ☐ XP practices based maintenance

How many hours your team has spent in writing/ modifying the code?

_________________(in hours)

Specify your role in maintenance project?

Problems faced during maintenance of a system, which is developed by others?

Rate your confidence that the code you wrote/ modified for maintenance project is correct.

- ☐ 1 - Very low
- ☐ 2 - Low
- ☐ 3 - Average
- ☐ 4 - High
- ☐ 5 - Very high

Rate your confidence that the code you wrote/ modified for maintenance project is easily comprehensible.

- ☐ 1 - Very low
- ☐ 2 - Low
- ☐ 3 - Average
- ☐ 4 - High
- ☐ 5 - Very high

Rate your confidence that anybody can make changes to the code without breaking it.

- ☐ 1 - Very low
- ☐ 2 - Low
- ☐ 3 - Average
- ☐ 4 - High
- ☐ 5 - Very high

Rate your confidence that anybody can easily reuse the code in another project.

- ☐ 1 - Very low
- ☐ 2 - Low
- ☐ 3 - Average
- ☐ 4 - High
- ☐ 5 - Very high

If you were given a choice of writing/ modifying code with traditional or XP practices based approach, which would you choose?

- ☐ Traditional
- ☐ XP practices based approach

Why above approach is selected?

Additional comments:

Thanks for sparing crucial time to fill the questionnaire.

Questionnaire for XPM Team Members

This part of final survey is conducted after completion of experiment that reveals experience of XP practices in maintenance. Tick mark in the choices or fill values in specified range, as the case may be.

What advantages and disadvantages you found after appling following XP practices during maintenance?

Pair programming

Advantage:

Disadvantage:

Refactoring

Advantage:

Disadvantage:

Test driven development

Advantage:

Disadvantage:

Continuous integration

Advantage:

Disadvantage:

On-site customers

Advantage:

Disadvantage:

Small releases

Advantage:

Disadvantage:

Planning game

Advantage:

Disadvantage:

Additional Comments:

Thank you for giving time to complete the questionnaire.