The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.21 e-ISSN: 2251-7545

Code Change Approach for Maintenance using XP Practices

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

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

***Abstract*- Software is developed with prior requirements and it is maintained continuously with rapid progresses in domain, technology, economy and other fields. The core activity of maintenance is code change, which changes the code to remove a bug or add new functionality. Maintenance projects contain an unstructured code due to patched and repatched software while addressing successive customer issues. Change in unstructured code without proper test coverage is a risky job. Software maintenance process slowdowns due to lack of proper test suite. Software maintenance process can also be affected due to staff turnover, low team morale, poor visibility, complexity of maintenance projects and lack of communication techniques among stakeholders. On the other hand, Extreme Programming (XP) practices such as Test Driven Development (TDD), refactoring, pair programming and collective ownership can overcome some of the challenges of maintenance up to some extent for non-XP projects. In this paper, an integrated code change approach is proposed for software maintenance using XP practices such as TDD, refactoring and pair programming. The proposed approach uses RC story, production code and test code of existing system during code change. The proposed approach is validated by applying it on several academic projects of software maintenance. It is observed that the proposed approach provides higher quality code in terms of the structure, correctness, robustness and maintainability hence improving software design. The XP practices based approach enhances both learning and productivity of the work by improving courage, team morale and confidence to support higher motivation in code change. In order to improve proposed approach, this experiment can be replicated in future to collect more data and to validate the observations.**

Keywords-Software maintenance; extreme programming; code change approach

I. INTRODUCTION

Software maintenance is the process of modifying a software product after delivery to correct faults or to implement new functional requirements. Software maintenance helps to improve performance, reliability, and adaptability for change request in the product in a modified environment. It is categorized as adaptive, corrective, preventive and perfective maintenance [1, 2]. The maintenance of legacy code is a tedious, expensive, and error prone task due to absence of test coverage, incomplete or out-of-date documentation and unavailability of original developer. It is hard to predict the impact of changes in legacy code due to its complex structure [3, 4]. Thus, change in the code without sufficient test coverage can result in system instability and bugs.

Extreme Programming (XP) is a software development methodology, which intends to improve software quality and responsiveness to changing customer requirements. XP is one of the important implementation of agile philosophy. It is a light-weight methodology for teams of approximately 10 people developing software in the face of vague or rapidly changing requirements [5]. XP is build upon various existing and common sense practices and principles, but applies these to extreme levels. For example, code review, testing, designing, and refactoring are preformed continuously, rather than at dedicated phases of the software process only.

Extreme programming practices are commonly used during software development with maintenance as its regular phase. But during maintenance of non-XP projects, XP practices can be applied by practitioners will require a dedicated process model for legacy system maintenance. The iterative maintenance life cycle using extreme programming is a process model for software maintenance that help to resolve the problems such as, unstructured code, team morale, poor visibility of the project, lack of communication techniques and lack of proper test in maintenance process [6]. It uses RC story as a requirement artifact, which can be written by the end users of software for maintenance [7]. RC stories provide end user collaboration and simplify requirement engineering process of software maintenance. The frequent tribulations such as, poor visibility of the project, lack of communication in maintenance process can be resolved by RC story format. The estimation of RC story of iteration is performed using SMEEM [8, 9]. However, it is still unidentified as to how much XP practices affect the structural quality parameter of code and interest toward maintenance activity during change implementation phase.

In order to investigate the effects of XP practices such as, TDD, refactoring and pair programming in change implementation for maintenance, a code change approach is proposed that perform investigation of differences between code change using XP practices and traditional approaches. To validate proposed approach, experiments were carried out, where the maintenance practitioners are asked to perform changes to an existing code by using proposed code change approach as well as traditional approach. The results

The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.21 e-ISSN: 2251-7545

of experiments of both approaches are compared on the basis of time duration and source code quality parameter. This approach will be helpful in incorporating changes in legacy code of different type of maintenance activities.

The rest of the paper is organized as follows; Section II discusses related work on XP practices used in software maintenance. The illustration of proposed approach of code change is provided in Section III. Section IV covers case study for validation of proposed technique. The results are discussed in Section V. Finally, the concluding remark and future work will be presented in Section VI.

II. RELATED WORK

There are several advantages of XP practices during software maintenance [4, 6, 12, 17]. XP makes extensive use of unit test [5, 10]. Automated unit test provides several advantages in maintenance process such as, instant feedback when working on a legacy code, confidence and courage to make error prone modifications, improve more code readability, and reduces duration of impact analysis before any modifications [11, 12, 13, 14, 15]. It is prerequisite that planned change for doing any refactoring be supported with test cases. It supports addition of new features and fixing bugs in a safe manner. During maintenance, profitably test cases are written, run, and passed for source codes. Test-driven maintenance will increase confidence in the code, reduces the risk of failure, speeds up development, and produces more robust code [16]. It is similar to the bottom-up program comprehension approach [17, 18]. However, adding test cases during maintenance of a complex legacy code is difficult as it slows down maintenance process. Unit tests themselves requires maintenance, i.e., extensive unit tests have to be also changed when production code is changed. Before introducing new code in existing legacy code, automated test suite should be written in initial iteration [14]. Writing unit tests for an entire large legacy system at once is time consuming and practically infeasible. To solve this problem, some prioritization criteria can be helpful such as, divide and conquer on the basis of function size, modification frequency and bug fixing frequency.

Refactoring is the process of altering existing software with series of particular transformations that improve the code without changing its behavior [19]. Refactoring removes unnecessary complexity thereby making change in code easier, faster and improves the structure of the code and hence, readability. In XP, all the tests have to be executed after refactoring [6, 20]. Refactoring can be used in top-down and bottom-up program comprehension [18]. Refactoring can be applied either by analyzing code for refactoring using bug density analyzed using bug tracking system or on encountering bad smells in the code [12]. Refactoring supports courage needed to make changes in legacy code and simplify designs [21]. The tests have to be at place for refactoring to be safe, due to this reason tests should be introduced to the code prior to refactoring [18, 5, 15, 22]. Writing unit tests and refactoring in small steps devise code testable [22, 23, 24]. Reverse and forward refactoring activity deals with two usually opposed program properties efficiency and understanding can be useful for maintenance [25]. With tests, changes can be performed in a better way.

Change operation of legacy code in maintenance projects is often mind-numbing. With pair programming, it might be more enjoyable, explore more alternatives, and work better on complex problems. Using this practice, knowledge is spread among the maintenance team using frequent partners' switching. Pairs provide better result on solving complex problems [26, 27]. They come up with better solutions that are easier to maintain later stages as decision are made by the pair. Coding standard is more strictly adhered with improved structure of code and code quality with 15% less defects [26, 27]. It is observed that 15% more time is consumed in pair programming but XP proponents claim that the time loss is regained because of the improved code quality.

Each of these practices i.e. TDD, refactoring and pair programming have their own advantages and limitation in the respective applications and areas. But there does not exist any standard approach that could integrate all three practices, i.e., TDD, refactoring and pair programming; and perform together for better results. In this paper, an integrated approach is proposed covering all three practices. Main phases of code change approach are requirement artifact extraction, check artifact availability, artifact extraction, test case creation or modification, create or modify production code, run test case and apply continuous integration. This approach changes legacy code in an iterative manner. The proposed approach is discussed in Section III with its phases. In this way, the proposed approach super shades the benefits of existing practices and provides a standard approach for code change to the maintenance projects.

III. CODE CHANGE APPROACH

The existing process models of software maintenance uses traditional approaches to change code according to the new requirement or to remove bugs. In this paper, a code change approach is proposed based on XP practices. The concept of proposed approach is shown in Fig. 1. The main aspects of this approach are artifact extraction from repository apply TDD, refactoring and pair programming for bug removal and new feature development.

It uses Request for Change (RC) stories, source code and test cases of existing software as input and performs all the phases in the proposed technique. The main phases of this approach are requirement artifact extraction, check artifact availability, artifact extraction, test case creation or modification, create or modify production code, run test case and apply continuous integration. These phases are performed in an orderly fashion to produce a better structural code with complete test coverage. The individual phases are illustrated in subsequent paragraphs.

The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.21 e-ISSN: 2251-7545

A. *Requiremnt Artifact Extraction*

In this phase a RC story from RC story data base is extracted. The RC story database is maintained by the Project Manager.

B. *Check Artifact Availability*

In this phase, checking the availability of artifacts is performed to find whether production code and test cases for a particular RC story are available. There might be the following three results of matching process; both test case and production code are not available, test case is not available and production code is available; and test case and production code both are available. The signature matching process can be used to check availability of test case and source code.

C. *Artifact Extraction*

After checking availability this phase, relevant test cases and production code are extracted from repository. Test case and production code repository is maintained with the help of existing system. For example, RC1, a requirement change story, which is fulfilled by two production code classes and respective test cases will be extracted from the repository. The repository contains test cases and production code of existing system and it is maintained during each and every change.

D. *Test Case Creation or Modification*

If test case and production code both are not available for a particular RC story then test case ca be written according to the requested features. If test case is not available and production code is available for a particular RC story then program comprehension can be applied to write test cases according to the structure of production code. If test case and production code both are available for a particular RC story then test case can be modified according to the requirement change. In initial iteration, emphasise should be given on writing test cases to obtain more and more coverage for legacy code, thereafter bug fixing and other maintenance activity are performed. Pair programming practice can be used here for test case creation.

E. *Create or Modify Production Code*

If test case is ready then pair programming can be applied to create or modify production code to pass tests. Here, refactoring process will be applied in small steps to incorporate new requirement change. Test case, which is already implemented, plays a vital role and provides confidence in this process of refactoring. For example, if r_1, r_2, $r_3 \ldots r_n$, are different steps in refactoring and v_1, v_2, $v_3 \ldots v_n$ are different versions of code then v_s is the version of code, which is the most suitable for change; where $1<=s<=n$. Performing refactoring in this phase also improves comprehensibility and maintainability of the code for future maintenance.

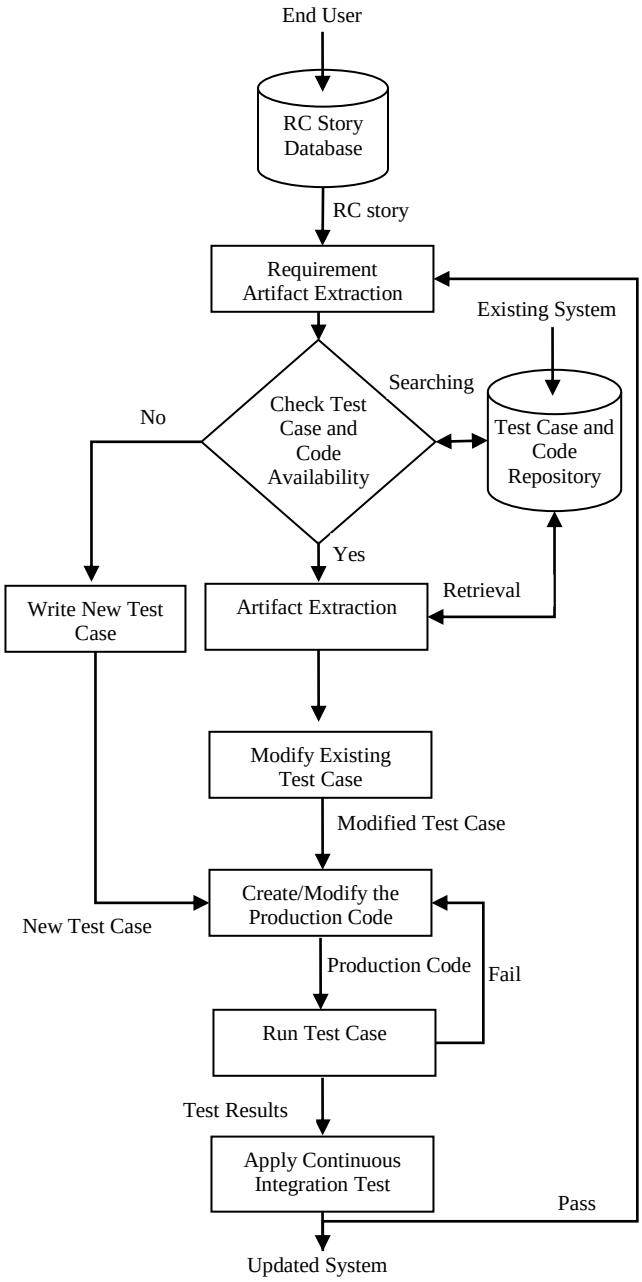

Figure 1. Code Change Approach Using XP Practices

F. *Run Test Case*

Test cases are executed to ensure that they are passing with modified or new production code. If test fails then production code can be modified and re-run the tests. If test case passes then the same approach can be used for other RC story of maintenance project. Thereafter, production code and test case are stored in the repository for future reference.

The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.21 e-ISSN: 2251-7545

Finally, integration is performed at the end of the day to integration with the existing code.

G. Apply Continious Integration Test

Here, continuous integration testing is applied to ensure that a change in the code for a RC story will not introduce new faults. Using this step in code change approach, we determine whether a code change in one part of the software affects other parts of the software. During this phase, we re-run previously run tests and check whether behavior of the program has change or not.

On successful execution of the phases of proposed code change approach, the quality of production code will improved with sufficient test coverage. The maintainability of the product in this way can be much better as compared to the product developed by existing approach. The approach can be more effective if required test cases and production codes are readily available in the repository. During pair programming; if one of the member of pair already involved in the existing system development; then it will ease the program comprehension and improve productivity. However, the approach is validated with all its phases with the help of case study, which is illustrated in the subsequent section.

IV. CASE STUDY

The case study for maintenance projects for this experiment involves three applications for the validation of proposed approach. Initially three applications were originally developed by three project groups of an Institute without using XP practices. All the three applications are currently in use in an academic environment. The overview of these projects is described in successive paragraphs with required maintenance task in the form of RC story and particularized description is presented in Table I.

TABLE I. PROJECT DESCRIPTION

Sr. No	Project Name	Project Category	Technology	Modules
A	Exam Control- room Management	Desktop Application	Java	Examination Superintendent
B	Student Feedback System	LAN based Application	J2EE	Student, College Management, Administrator
C	Library Circulation System	LAN based Application	Perl	Student, Liberian, Administrator

Project-A: Exam Control- room Management, ECM is a desktop application used to provide examination control room facility to the superintendent of an examination center. The superintendent can perform control room activity through application such as preparing requisition of material for exam, checking student detail of eligibility for exam, to prepare duty chart and seating plan, to prepare dispatch formats according to the university standards. The following RC stories are requested for maintenance.

RC story 1: Superintendent can maintain answer book record.

RC story 2: View the previous exam record in the specific format.

Project-B: Student Feedback System, SFS is a LAN based student feedback system for an Institute. Students can register and submit feedback of teacher through web application running on LAN. After successful submission of feedback, college management can view the feedback in different format and can generate reports for different purpose. The following RC stories are requested for maintenance.

RC story 1: Student can submit feedback on the basis of subject code.

RC story 2: College management can view and print department wise feedback reports.

Project-C: Library Circulation System, LBS is developed in Perl and MySQL. Students use their respective login to maintain profile, view issued book detail, apply reserve for book. Librarian uses their login to issue book, book return, view student and book detail, generate various reports etc. The following RC stories are requested for maintenance.

RC story 1: Calculation of fine under different heads.

RC story 2: Start SMS alerts.

The end users of applications require some correction and need to include new features such that information is available in more efficient and convenient manner. The requirement change is written by end user in the form of RC story. In this case study, two RC stories are considered from each project. The maintenance work of above projects were assigned to the three new project groups, each project group have three postgraduate student members. In each project group, two students form one pair and the rest one is required to work individually. Pair and individual both solve same RC stories and work under the same conditions. Pair members of a group apply XP based approach for code change whereas individual member apply existing approach for code change. Implementation of RC stories is performed in incremental order. Students have some experience of languages in which their respective application was developed. They have never performed XP practices before, nor do they have experience of maintenance projects. For observation code, snap-shots and voice-recording are considered. The data pertaining to the observations on these projects are discussed in Section V.

V. EVALUATION OF SOFTWARE QUALITY PARAMETERS

All three groups of maintenance projects were completed their tasks. Maintainers were interviewed and the workings of systems under maintenance were checked. During interview of maintainers, they were interviewed with defined questions on the basis of certain parameters such as courage

The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.21 e-ISSN: 2251-7545

and confidence, understanding of system, interest in maintenance activity etc. After completion of the tasks, the code, snapshot and voice recording were observed and checked on some parameters for each group such as, time taken in completing each RC story etc. It is shown in Table II, which is pictorially represented in Fig. 2. The structure of the code, number of lines of code written for a RC story by programmers, number of defects in the code, number of necessary and unnecessary change made in classes during change propagation are also observed. Results of experiment indicate that proposed approach produced better quality codes in terms of extendibility, understandability, comprehensibility etc. as compare to the existing approach, as shown in Table III and Fig. 3.

TABLE II. COMPARISON OF TIME (HOURS) CONSUMED IN BOTH APPROACHES

S. No.	Maintenance Task	Existing code change approach (in hours)			Proposed code change approach using XP (in hours)		
		A	*B*	*C*	*A*	*B*	*C*
1.	RC story 1	13	8	9	10	7	7
2.	RC story 2	10	6	8	6	4	5

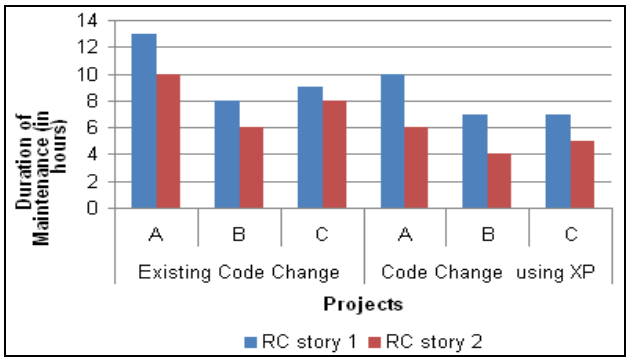

Figure 2. Comparison of time (hours) consumed in both approaches.

TABLE III. COMPARISON OF QUALITY PARAMETERS IN BOTH APPROACHES

S. No.	Name of Parameter	Existing code change approach (in %)			Proposed code change approach using XP (in %)		
		A	*B*	*C*	*A*	*B*	*C*
1.	Extendibility	30	60	50	70	75	70
2.	Understandability	55	55	60	80	80	68
3.	Reusability	45	50	60	90	85	88
4.	Efficiency of code	70	75	70	80	80	80
5.	Integrate-ability	75	70	80	75	80	80
6.	Maintainability	60	70	70	65	85	90
7.	Comprehensibility	60	60	65	80	80	75
8.	Testability	40	60	80	80	60	90
9.	Reliability	70	60	70	75	90	80
10.	Robustness	60	70	70	70	85	80

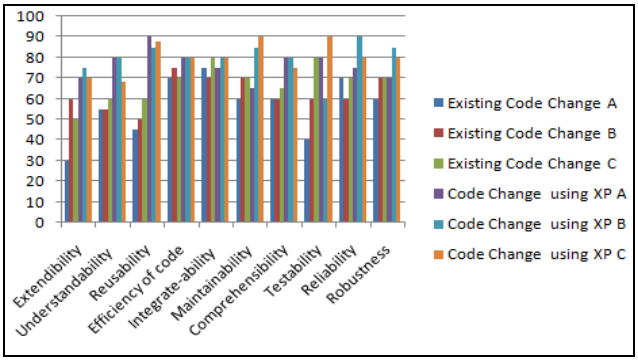

Figure 3. Comparison of quality parameters in both approaches.

VI. DISCUSSION

Some of the interesting facts have been observed regarding proposed approach in the view of maintenance practitioners and enhance product features. XP based approach enhances learning and speeds up the work by improving courage, team morale and confidence to support higher motivation in code change. It improves interest in maintenance activity through sharing of ideas using pair programming. Pair proposes better alternative solutions and understanding towards code change as compared to individuals. In proposed approach, comprehension activity requires less time as compared to existing approach. Using proposed approach, maintenance practitioners gain better understanding of overall system, which is shown in Table II. It is observed the RC stories 2 in all projects were implemented in less time duration. As the project progresses, the pairs use their experiences for better solution and understanding. Code change approach using XP practices provides higher quality code in terms of the structure, correctness, robustness and maintainability of code; hence, improving software design. Change propagation task is performed more correctly in proposed approach. The proposed approach generates code and test classes that can be reused by multiple applications as they are having well structured and generalized for common applications. The code and test classes generated by the proposed approach are self-documented.

VII. CONCLUSIONS AND FUTURE SCOPE

The maintenance of legacy code is a tedious, expensive, and error prone task due to absence of test coverage, incomplete or out of date documentation and unavailability of original developer. To study the affect of XP practices on structural quality parameter of code and interest towards

Doi: 10.7321/jscse.v3.n3.21 e-ISSN: 2251-7545

maintenance activity during change implementation phase, here, a code change approach of maintenance using XP practices is proposed. To validate investigation results, with case studies experiment were performed, where the maintenance practitioners were asked to make changes to an existing code by using both, proposed and traditional code change approach. We have compared the results of experiments on the basis of some code structural quality parameters. After experiment, it is observed that code change approach using XP practices provides higher quality code in terms of the structure, correctness, robustness and maintainability of code; hence, improving overall design of software. XP practices based approach enhances both learning and productivity of the work by improving courage, team morale and confidence to support higher motivation in code change. The observations of the proposed approach can be more effective and experimental by applying it on large projects with more team members. Future work will focus on replicating this experiment in order to collect more data and to validate our observations.

REFERENCES

[1] B. P. Lientz and B. E. Swanson, Software maintenance management, Addison- Wesley publishing company, 1980.

[2] A. Behforooz, and F.J. Hudson, Software Engineering Fundamentals. Oxford University Press, pp. 382–392, 1996.

[3] V. Rajlich, "Comprehension and Evolution of Legacy Software," In Proceedings of the 19th International Conference on Software Engineering, Boston, USA, pp. 669-670,1997.

[4] A. van Deursen, T. Kuipers, and L. Moonen, "Legacy to the Extreme,". In M. Marchesi and G. Succi, editors, Proceedings of the 1st International Conference on eXtreme Programming and Flexible Processes in Software Engineering - XP2000.

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

[6] J. Choudhari and U. Suman, "Iterative Maintenance Life Cycle Using eXtreme Programming," Proc. International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom-2010), IEEE Computer Society,October 15 - 16, 2010, pp. 401 – 403.

[7] J. Choudhari and U. Suman, "Designing RC Story for Software Maintenance and Evolution," Journal of Software (JSW), Academy Publisher, vol. 7, no. 5, May 2012, pp. 1103-1108.

[8] J. Choudhari and U. Suman, "Story Points Based Effort Estimation Model for Software Maintenance," Proc. 2nd International Conference on Computer, Communication, Control and Information Technology (C3IT- 2012), ScienceDirect by ELSEVIER, February 25 - 26, 2012, pp.761-765.

[9] J. Choudhari and U. Suman, "Phase wise Effort Estimation for Software Maintenance: An Extended SMEEM Model," Proc. CUBE International IT Conference & Exhibition, ACM Digital Library, September 3 - 5, 2012, pp.397-402.

[10] K. Beck, "Embracing Change With Extreme Programming," IEEE Computer, Vol. 32 Issue 10, Oct. 1999, pp. 70-77.

[11] P. Schuh, "Recovery, Redemption, and Extreme Programming," IEEE Software, Volume: 18 Issue: 6, Nov/Dec 2001, pp. 34-41.

[12] C. Poole, and J.W. Huisman, "Using Extreme Programming in a Maintenance Environment," IEEE Software, Vol. 18 Issue 6, Nov/Dec 2001, pp. 42-50.

[13] G. Wright, "eXtreme Programming In A Hostile Environment," Proc. Third International Conference on Extreme Programming and Flexible Processes in Software Engineering - XP2002, pp. 48-51.

[14] E.C. Eckman III, "XP Transition Roadmap," Proc. Third International Conference on Extreme Programming and Flexible Processes in Software Engineering - XP2002, pp. 219-222.

[15] A. Jalis, "Probe Tests: A Strategy for Growing Automated Tests around Legacy Code," In D.Wells and L.A.Williams (Eds.): Extreme Programming and Agile Methods - XP/Agile Universe 2002, pp.122-130.

[16] Elliotte Rusty Harold. "Testing legacy code." available at http://www.ibm.com/developerworks/java/library/j-legacytest/index.html.

[17] A. van Deursen, " Program Comprehension Risks and Opportunities in Extreme Programming," Proc. 8th Working Conference on Reverse Engineering, (WCRE'2001), IEEE Computer Society, pp. 176-185.

[18] A. van Deursen, L. Moonen, A. van den Bergh, and G. Kok, "Refactoring Test Code," Proc. 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2001), M. Marchesi and G. Succi (Eds.), 2001.

[19] M. Fowler, "Refactoring: Doing Design After the Program Runs," Distributed Computing, Sept. 1998, pp. 55-56.

[20] M. Feathers, "Working Effectively With Legacy Code" available at http://www.objectmentor.com/resources/articles /WorkingEffectivelyWithLegacyCode.pdf.

[21] L.Tokuda, and D. Batory, "Evolving Object-Oriented Designs with Refactorings," Proc. 14th International Conference on Automated Software Engineering, IEEE, pp. 174-181.

[22] S. Freeman and P. Simmons, "Retrofitting Unit Tests", Proc. Third International Conference on Extreme Programming and Flexible Processes in Software Engineering - XP2002, pp. 11-15.

[23] F. Ricca, M. Di Penta, M. Torchiano, "Guidelines on the use of Fit tables in Software Maintenance Tasks: Lessons Learned from 8 Experiments".

[24] T. Bhat and N. Nagappan, "Evaluating the Efficacy of Test-Driven Development: Industrial Case Studies".

[25] G. Villavicencio, "Software Maintenance Supported by Refactoring".

[26] A. Cockburn, and L. Williams, "The Costs and Benefits of Pair Programming," In M. Marchesi and G. Succi, editors, Proc. 1st International Conference on eXtreme Programming and Flexible Processes in Software Engineering - XP2000, 2000.

[27] L. Williams, R.R. Kessler, W. Cunningham, and R. Jeffries, "Strengthening the Case for Pair Programming," IEEE Software, vol. 17, issue: 4, Jul/Aug 2000, pp.19-25.