CHAPTER 5
CODE CHANGE APPROACH USING XP PRACTICES

5.1 INTRODUCTION
Change implementation is a crucial and central phase in iterative maintenance life cycle using XP model. This phase is involved for changes in source code to remove a bug, change existing functionality or adding a new functionality in software system. Performing changes in legacy code of software system are very difficult and involves risk due to unstructured code and unavailability of the test coverage. Changes in legacy code is a tedious, expensive, and error prone task due to the absence of test coverage, incomplete or outdated documentation and unavailability of original developers. The impact of changes is hard to predict in legacy code due to its complex structure. Thus, changes in the code without sufficient test coverage may result into system instability and presence of bugs. It is observed that the changes in the existing code is affected due to staff turnover, low team morale, poor visibility, complexity of maintenance projects and lack of communication among maintainers.

Coding is the central point of extreme programming. The development practices of extreme programming, i.e., test-driven development (TDD), refactoring, pair programming, and the continuous integration are significant in iterative software development. These practices help in producing clean and maintainable source code in the new development. Research shows that the development practices of XP have capability to deal with the aforesaid problems of legacy code in a software system [35].

Literatures reveal that the test-driven development practice of XP can provide support for software maintenance. Test-driven
development produces extensive test coverage that provides instant feedback while working with legacy code [18, 47, 48]. It can provide courage and confidence to team members while performing error-prone modifications [49, 50, 51]. Duration of the impact analysis can be reduced substantially by extensive test coverage. Extensive test coverage in TDD supports refactoring through its test cases. Addition of new features and fixing of bugs through TDD provide even better results. Development of test coverage for large legacy code are difficult and time consuming task. The development of unit tests for a large legacy system as a whole is not feasible. Therefore, the use of prioritization criteria, viz., divide and conquer on the basis of function size, modification frequency and bug fixing frequency can be helpful to solve the aforesaid problems [25]. The test coverage, which is achieved using TDD in initial iterations of maintenance, can improve the understandability and readability of source code for future maintenance.

Another development practice of XP is refactoring, which provides a set of activities that transforms the design of source code without affecting its existing behavior [53]. Refactoring removes unnecessary complexity, thereby making changes to the code easier and faster [25]. It also improves the structure of code and hence, increases its extensibility and readability. Refactoring can be applied using bug density analysis or on encountering bad smells in the code [18, 55]. Refactoring process requires courage to perform changes in the legacy code for design simplification [27]. Back-up of test coverage provided by TDD is necessary for an effective and efficient refactoring. In XP, tests are executed after refactoring to ensure that code behaviors is intact after refactoring [35, 54]. For this reason, test cases should be written for the code prior to refactoring [22, 50, 52, 56]. Also, writing unit tests and applying reverse and forward refactoring in small steps help to ease maintenance task [56, 57, 58]. Automated tools, for example, eclipse supports unit testing along with refactoring [96].
In pair programming, two programmers work side-by-side on design, implementation, and testing with a single computer. It is collaborative approach, where one programmer works as driver who writes the code and another is the observer who performs the code review. Pair programming shows its significance during maintenance. Flor and Hutchins [94] has conducted an experiment on maintenance project and reported that the pair programmers improve the software design. Xu and Chen [95] has performed an academic experiment on a maintenance project and conclude that the pairs programmers provide better solution for urgent maintenance tasks as compared to individual. They also reported that programmers, when working as pair sharing ideas, performs quality control with enhanced learning and improved speed of the work during software evolution. Programming pairs are more willing to perform a broader search to understand the program. Also, they have higher motivation to finish the tasks than individual programmers. Thus, pair programmers contribute to the time difference on the tasks and perform change operations of legacy code more enjoyable. The solution of a complex problem during maintenance in pair explores more alternative solutions [25]. Using this practice, knowledge is spread among the maintenance team using frequent partner switching [28, 29]. The partner switching concept of pair programming and the collective code ownership practice can mitigate the problems arising from staff turnover. Continuous integration practice of XP can be used for instant feedback during the maintenance [25].

Each of these practices, i.e., TDD, refactoring, pair programming, and continuous integration have their own advantages and challenges for maintenance. But there does not exist any standard code change approach that could integrate these development practices to perform together as a whole for better maintenance results. The existing process models of software maintenance use traditional approaches towards code change for the new requirements or bugs removal. Therefore, there is a need for a systematic code change approach in change
implementation phase of the iterative maintenance life cycle using XP. In this direction, we have proposed an integrated approach for code change that covers four development practices of XP (i.e. TDD, refactoring, pair programming, and continuous integration). Main phases of the proposed code change approach are artifact extraction, artifact availability check, source code extraction, test case creation or modification, create or modify production code, test case execution, and continuous integration. This approach changes legacy code in an iterative manner. To validate proposed approach, experiments were carried out, where the maintenance was performed to change existing codes by using proposed code change approach as well as traditional approach. The results of experiments of both approaches are compared on the basis of time duration and source code quality parameters. This approach will be helpful in incorporating changes in legacy code of different type of maintenance activities.

The proposed approach supershades the benefits of existing practices and provides a standard approach for code change in maintenance projects. Rest of the chapter is organized as follows. The proposed approach for code change is illustrated in Section 5.2. Section 5.3 presents a case study for validation of proposed technique. Productivity evaluation is performed in Section 5.4. Code quality evaluation is performed in Section 5.5. Observations of experiment are discussed in Section 5.6. Lastly, Section 5.7 describes the summary of presented research.

5.2 CODE CHANGE APPROACH

Code change approach is a systematic process for producing high quality code. It applies development practices of XP while changing the legacy code. It is the core activity in change implementation phase of iterative maintenance life cycle using XP. The process of proposed approach is shown in Figure 5.1. The main aspects of this approach are artifact extraction from repository and applying pair programming.
Figure 5.1: Code Change Approach using XP Practices
refactoring, TDD, and continuous integration for bug removal and new feature development. It uses RC stories, updated design documents, source code, and test cases of existing software system as input and performs all the phases of the proposed approach. The main phases of this approach are artifact extraction, artifact availability check, source code extraction, test case creation or modification, create or modify production code, test case execution and continuous integration. These phases are performed in an orderly manner to produce a better structural code with complete test coverage. The individual phases are illustrated in subsequent subsections.

5.2.1 Artifact Extraction

In this phase, requirement artifact, i.e., RC stories and related tasks are extracted from the RC story database maintained by the project manager. The updated design documents are also extracted from the repository of design document. During iteration planning, a single RC story is divided into multiple tasks and it is assigned among different pairs of maintenance team. Design documents are updated and maintained during the design revision phase of the iterative maintenance life cycle using XP.

5.2.2 Check Source Code Availability

In this phase, source code availability is checked to find whether production code and test cases for a particular task of an RC story are available or not. There may be three possibilities for a maintenance task; both test case and production code are not available, test case is not available and production code is available; and both test case and production code are available. The signature matching process can be used to check availability of test case and source code.

5.2.3 Source Code Extraction

After checking the availability of source code, relevant test cases and production code are extracted from the code repository. Test case
and production code repository is maintained with the help of existing system. For example, a change requirement 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 for each and every change.

5.2.4 Test Case Creation or Modification

The unavailability of test case and production code for a task of an RC story require test case to be written according to the requested features. If test case is unavailable and production code is available for a particular maintenance task then program comprehension can be applied to write test cases according to the structure of production code. On availability of both, test case and production code, for a particular maintenance task lead to test case modification according to the requirement change. In initial iteration, emphasis should be given on writing test cases to obtain more and more coverage for legacy code, thereafter bug fixing and other maintenance activities are performed. Pair programming practice can be used here for test case creation. Tools that support automated unit testing such as Eclipse IDE can be used for test code creation.

5.2.5 Create or Modify Production Code

This phase leads to the modification of existing code or creating a fresh code according to test cases obtained from previous phase. If test case is ready then pair programming can be applied to create or modify the production code to pass the unit tests. Here, refactoring process is applied in small steps to incorporate new requirement change. Test cases already implemented in previous phase play a vital role and provide confidence to the 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 most suitable for change; where, \( 1 \leq s \leq n \). Refactoring practice also improves comprehensibility and maintainability of code for future maintenance.
Tools that support automated unit testing along with refactoring such as Eclipse IDE can increase productivity in this phase.

5.2.6 Test Case Execution
Test cases are executed to ensure that they are passing the modified or new production code. If test fails then production code can be modified and re-runs the tests. If test case passes then the same approach can be used for other task of an RC story. Thereafter, production code and test case are stored in the repository for future reference. Finally, integration is performed at the end of the day to integrate changes in the existing code.

5.2.7 Continuous Integration Test
Continuous integration testing ensures that a change in the code for an RC story will not introduce new faults. This step in code change approach determines whether code change in one part affects other parts of the software. During this phase, we re-run previously run tests and check whether behavior of the program has changed or not. Compatibility issues and interfacing problems of maintenance can resolve by the continuous integration and testing practice. Automated tool such as, TeamCity, Bamboo, and CruiseControl can be used for continuous integration.

The successful execution of different phases of proposed code change approach improves quality of production code with sufficient test coverage thereby increasing maintainability of code. This 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 members of pairs were involved during the existing system development then it becomes easy to perform program comprehension for improving productivity. However, the approach can be validated with the help of case study, which is illustrated in the subsequent section.
5.3 CASE STUDY

The case study of maintenance projects for this experiment involves three applications for validating proposed approach. These applications are Exam Control Room Management, Student Feedback System and Library Circulation System. These applications were originally developed by earlier groups of students of an institute without using XP practices. These applications are used to handle the academic activities of the institute. The overview of these applications is discussed in successive paragraphs with required maintenance tasks in the form of RC stories and particularized description is presented in Table 5.1.

Table 5.1: Project Description

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Project Name</th>
<th>Project Category</th>
<th>Technology</th>
<th>Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Exam Control-room Management</td>
<td>Desktop Application</td>
<td>Java</td>
<td>Examination Superintendent</td>
</tr>
<tr>
<td>2.</td>
<td>Student Feedback System</td>
<td>LAN based Application</td>
<td>J2EE</td>
<td>Student, College Management, Administrator</td>
</tr>
<tr>
<td>3.</td>
<td>Library Circulation System</td>
<td>LAN based Application</td>
<td>Perl</td>
<td>Student, Librarian, Administrator</td>
</tr>
</tbody>
</table>

Exam Control Room Management (ECM): It is a desktop application used in the examination control room. The superintendent of examination can control examination activities through this application such as, preparation of requisition of materials for an exam, checking students' details of their eligibility for appearing in the exam,
preparation of invigilator’s duty chart, seating plan, preparation of dispatch formats according to the university standards and so on. The following RC stories were submitted by end users of the system.

RC story 1: Superintendent can maintain records of answer books.
RC story 2: View the records of previous exams in a specific format.

**Student Feedback System (SFS)**: It is a LAN based student feedback system for an institute. Students can register and submit feedback about teachers through a web application running on the LAN. After successful submission of feedback, college management can view the feedback in different formats and can generate reports for different purposes. The following RC stories were submitted by end users of the system.

RC story 1: Student can submit feedback on the basis of subject code.
RC story 2: College Management can view and print a department-wise feedback report.

**Library Circulation System (LBS)**: It is developed in Perl and MySQL. Students use their library accounts to maintain their profiles, view details of issued books, and apply reserve for a book. Librarians use their accounts to issue/return books, view student and book details, generate various reports etc. The following RC stories were submitted by end users of the system.

RC story 1: Calculation of overdue charges 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 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 applies 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 5.5.

5.4 PRODUCTIVITY EVALUATION

All the groups of maintenance projects completed their tasks. Maintainers were interviewed and the working of systems under maintenance was inspected. During interview of maintainers, various defined questions were asked on the basis of certain parameters such as, courage and confidence, understanding of the system, interest in maintenance activity etc. After completion of the tasks, the code, snapshot and voice recording were observed for each group. For example, time taken in completing each RC story. The observations i.e., time consumed in traditional as well as in XP based approach are shown in Table 5.2, which are pictorially represented in Figure 5.2. The structure of code, number of lines of code written for a RC story by programmers, number of defects in the code, number of necessary and unnecessary changes made in classes during change propagation etc., were also observed.

5.5 CODE QUALITY EVALUATION

An external code quality evaluation is conducted after the completion of code change. The evaluation sheet is presented in the
Table 5.2: Comparison of Time (in hours) Consumed in Both Approaches

<table>
<thead>
<tr>
<th>Maintenance Task</th>
<th>Existing Code Change Approach (in hours)</th>
<th>Proposed Code Change approach using XP (in hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ECM</td>
<td>SFS</td>
</tr>
<tr>
<td>RC story 1</td>
<td>13</td>
<td>8</td>
</tr>
<tr>
<td>RC story 2</td>
<td>10</td>
<td>6</td>
</tr>
</tbody>
</table>

Figure 5.2: Comparison of Time (in hours) Consumed in Both Approaches
Appendix B-1. Experienced teachers and professional developers performed the evaluation of XP and Non-XP versions of maintenance projects. Reviewers spent three hours for the valuation of each version of projects. A briefing was provided to each reviewer about the project. The reviewers were selected based upon the teaching or development experience in the respective technologies. The reviewers were requested to provide a single value on a scale of 1 to 5. Table 5.3 presents the comparative values of various code quality parameters. The comparative chart for code quality in both approaches is shown in Figure 5.3. It is observed that the proposed approach has produced better quality code in terms of maintainability, extensibility, reusability, understandability, integration, testability, and robustness as compared to the existing approach. For example, in LBS project, average value of maintainability for code change approach using XP is 4.2 whereas the value of existing code change approach is 3.0.

5.6 DISCUSSION

Some of the interesting facts have been observed regarding proposed approach with respect to maintenance practitioners and enhanced product features. XP based approach enhances learning and speeds up the work by improving courage, team morale and confidence by supporting higher motivation during 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 can be observed through Table 5.2. Observations reveal that RC stories 2 in all projects were implemented in short time duration. As the project progressed, the pairs used their experiences for better solutions with more understanding. Code change approach using XP practices provides higher quality code in terms of structure, correctness,
robustness and maintainability of code; hence, improving the 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 well structured and are generalized for common applications. The code and test classes generated by the proposed approach are self-documented.

5.7 SUMMARY

Change in the source code is a very usual activity during maintenance to remove a bug or to change the existing functionality or to add new functionality in the software system. During maintenance, changing the legacy code is a tedious, expensive, and error prone task due to the absence of test coverage, incomplete or out of date documentation and unavailability of original developers.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Existing Code Change Approach</th>
<th>Code Change Approach using XP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ECM SFS LBS</td>
<td>ECM SFS LBS</td>
</tr>
<tr>
<td>Maintainability</td>
<td>2.5 3.5 3.0</td>
<td>4.0 3.5 4.2</td>
</tr>
<tr>
<td>Extensibility</td>
<td>3.2 3.8 2.5</td>
<td>4.7 4.8 4.5</td>
</tr>
<tr>
<td>Reusability</td>
<td>3.2 2.5 3.0</td>
<td>3.5 4.2 4.0</td>
</tr>
<tr>
<td>Understandability</td>
<td>3.0 3.5 2.0</td>
<td>4.5 3.5 3.5</td>
</tr>
<tr>
<td>Integration</td>
<td>3.5 3.0 2.5</td>
<td>3.2 4.5 3.5</td>
</tr>
<tr>
<td>Testability</td>
<td>2.0 2.5 3.0</td>
<td>4.5 4.8 4.2</td>
</tr>
<tr>
<td>Robustness</td>
<td>2.4 3.5 3.5</td>
<td>4.0 4.5 3.5</td>
</tr>
</tbody>
</table>

Table 5.3: Comparative Values of Code Quality Parameters
Software maintenance process can also be affected due to staff turnover and low team morale. On the other hand, development practices of XP, i.e., test driven development, refactoring, pair programming, and continuous integration help to resolve the aforesaid problems. To study the affect of XP practices on code quality and interest of maintainers towards maintenance activity during change implementation phase, in this study, a code change approach of maintenance using XP practices is proposed. The proposed approach uses RC stories, updated design documents, source code, and test cases of existing software system as input and performs code changes. The main phases of this approach are artifact extraction, check artifact availability, source code extraction, test case creation or modification, create or modify production code, test case execution and apply continuous integration.
The proposed approach is validated by applying it on academic projects of software maintenance. In this experiment, maintenance practitioners were asked to make changes to an existing code by using both, proposed and traditional code change approach. After the experiment, code produced in both approaches, was evaluated on the basis of code quality parameters. After the evaluation, results were compared. The controlled experiment reveals that code change approach using XP practices provides higher quality code in terms of maintainability, extensibility, reusability, understandability, integrate-ability, testability, and robustness. The proposed code change approach enhances both learning and productivity of the work by improving courage, team morale and confidence to support higher motivation in code change.