CHAPTER 8
CONCLUDING REMARKS

The thesis covers research work on investigation of extreme programming practices in software maintenance. It involves various aspects of software maintenance using XP such as, process model, change requirements elicitation, maintenance estimation, changes in legacy code, and maintainability measurement. In addition, it includes experimental evaluation and comparison of XP based process model with other process models of maintenance. The conclusion of the presented research is summarized in the following paragraphs.

The dedicated process model for software maintenance based on XP practices can help to improve quality of software product. Maintenance appears as an iterative development round of software development from outside. A closer study reveals the differences in the procedures emphasized during maintenance from those applied in the development life cycle of software. During maintenance, more effort is required on the early stages than the development; hence, it is mandatory to have maintenance-conscious models. Different maintenance-conscious models are quick-fix, Boehm, Osborne, iterative-enhancement, full-reuse etc. The existing software maintenance models are generally derived from the classic waterfall life cycle and other traditional software development models. The existing maintenance process models have limitations while handling critical problems such as, unstructured code, team morale, poor visibility, and lack of communication among stakeholders. The literatures show that XP practices applied to software maintenance projects provide better software solutions. During the maintenance of non XP projects, practitioners require a dedicated process model based on XP practices for legacy system maintenance. Therefore, a process
model for software maintenance, i.e., iterative maintenance life cycle using extreme programming has been presented in Chapter 2.

The proposed model has mainly seven phases; namely, identification and categorization, planning, analysis, design revision, change implementation, acceptance testing, and release. The proposed model incorporates XP practices in the different activities of phases to resolve the aforesaid problems of maintenance.

Problem reporting and elicitation of change requirements is an important part of software maintenance process. An unambiguous change requirement during maintenance reduces risks and contradictions among customers and maintainers. There exist very few artifacts for elicitation and documentation of change requirements. Change requirements in software maintenance are elicited and documented using existing artifacts such as, Software Problem Reports (SPRs) and Software Change Request (SCR). The SPR and SCR are unable to handle the problems of poor visibility and lack of communication between stakeholders. In SPR and SCR, there is very limited involvement of the customers, which is very crucial part in XP approach. The existing artifacts have certain limitations for the extreme programming based maintenance process. On the other hand, fields of user story are not sufficient from maintenance perspectives. Therefore, there is a need of dedicated change requirement artifact for the iterative maintenance life cycle using XP. We have designed a change requirement artifact, Request for Change (RC) story, which is described in Chapter 3. The proposed RC story format is based on user story of XP and SPR form of maintenance.

The proposed RC story is written by the customer for bug reporting, enhance requests or for the new requirements. The RC story improves customer collaboration and simplifies requirement engineering process of software maintenance. As the daily stand-up meeting is performed on the story board in the XP based maintenance
approach; therefore, the frequent tribulations such as, poor visibility of the project, and lack of communication in maintenance process can be resolved using proposed RC story format.

Project estimation plays a vital role in the success of software process. As software maintenance is an important activity and accounts for the majority of the software total cost but the researchers have less attention on maintenance estimation as compared to the estimation of new software development. There exist fewer effort estimation models for software maintenance as compared to software development. Software maintenance effort can be estimated by existing estimation models such as, ACT model, FP model, and COCOMO 2.0 reuse model. The different estimation models considers SLOC, FP and object points as sizing units for determining maintenance effort. The existing models of software maintenance estimation are based on the traditional software development methodologies. These models produce realistic results for similar methodology because such models consider FP, SLOC or object points as the basis of size measurement, which are not suitable for extreme programming based maintenance methodology. The non-algorithmic techniques of agile for the estimation such as, expert opinion, analogy, disaggregation, and planning poker are unpredictable in the absence of past data and experts. These techniques are used to derive the estimates on the basis of opinion of experts and historical data. Further, these methods may generate different estimates for same project depending on the intuition of the estimators. Therefore, there is a need of algorithmic approach for estimation in the iterative maintenance life cycle using XP. We have proposed an algorithmic estimation model, Software Maintenance Effort Estimation Model (SMEEM) in Chapter 4.

SMEEM uses story points as a size measure. The SMEEM model incorporates value adjustment factors for estimation of size and effort in a maintenance project. It is designed to help the maintenance
manager to calculate the estimated software maintenance effort in terms of cost, size and duration.

Change implementation is a crucial and central phase in the proposed XP based model. In this phase, changes are performed 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, change in the code without sufficient test coverage may result into system instability and bugs. Development practices of XP, 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 results. 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 in Chapter 5. The proposed code change integrates 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 performs changes in legacy code in an iterative manner. The proposed approach supershades the benefits of existing practices and provides a standard approach for code change in maintenance projects.
The cost of maintaining a software product and its lifetime is highly influenced by the maintainability. Maintainability is a critical issue for the software developed through non-XP methodology with legacy and unstructured code. Therefore, it is important to perform an experimental evaluation to observe the effect of iterative maintenance life cycle using XP on maintainability and productivity and other issues of maintenance. There are limited empirical evidences to show the effects of XP practices on maintainability and productivity during maintenance. Therefore, a controlled experiment is performed in Chapter 6. This experiment examines the effects of proposed XP based model on maintainability and productivity during maintenance. The experiment was conducted with postgraduate students. Students in two groups completed a semester on maintenance projects using either by iterative maintenance life cycle using XP or traditional maintenance approach.

In this experiment, we have used internal quality metrics for maintainability measurement and the comparison of maintenance models. This experiment demonstrates that the XP based model produces higher maintainable code as compared to traditional approach of maintenance. Experimental results also show that XP practices during maintenance enhance productivity by improving learning, team morale and confidence.

There are various process models for maintenance, which are derived from traditional models of software developments such as, quick-fix, Boehm, Osborne, iterative-enhancement, full-reuse etc. The process, phases, activities and artifacts in these models are different than the XP based maintenance model. Therefore, the comparative analysis of XP based model is performed with traditional process models in Chapter 7. The comparison is performed on the basis of various parameters such as, origin, requirement artifacts used, role of customer, planning, number of iterations, average duration of iterations, estimation techniques used, design improvement practices
used, programming style used, code ownership, build time, code review practices used, testing strategy applied, way of regression testing, amount of documentation, phase execution, briefing style used, application of process model etc. The comparative analysis of XP based model with traditional process models shows that XP based model is better approach than other models because it elicits maintenance needs using RC story that provides unambiguous change requirements with acceptance criteria. The RC story improves customer collaboration and resolves the problems of poor visibility of the project, lack of communication in maintenance process.

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 provides more realistic effort estimation results. XP practices based code change approach improves the maintainability of software product through the design improvement practices. Continuous integration and testing verify the impact of modifications that provides instant feedback. Besides that the proposed process model is an extreme programming based approach; it includes documentation for future maintenance activity. The iterative maintenance life cycle using XP consists of all features of a process model that can resolve problems of maintenance.

We have contributed towards solving the problems of software maintenance by investigating the use of XP practices including dedicated process model for maintenance based on XP practices, artifact for elicitation of change requirements, story points based estimation model, standard code change approach, evaluation of process model on the basis of maintainability and productivity, and comparative analysis with the effectiveness of XP based model.
Our contribution opens various further possible research areas in the domain of software maintenance using extreme programming. The research work on extreme programming in software maintenance can be extended in the following areas:

- Reuse oriented approach can be designed for software maintenance using extreme Programming.

- Application levels of XP practices such as, refactoring and TDD is also a research issue in maintenance.

- Another important research issue can be the empirical analysis for future of maintenance, on the basis of different deliverables and artifacts, which are produced in XP based maintenance process.

- An automated framework can be designed for software maintenance using XP that can support maintenance activities as well as XP practices.

- Configuration management and project management during XP based maintenance are also important areas of research.

- Selection of design mechanism is also a research issue in XP based maintenance environment.

- Design and selection of testing strategy for XP based maintenance environment and the impact of XP practices on the regression testing.

- Maintenance of open source software through XP is also an important area of research.

- Improvement in the test case design process using refactoring during preventative maintenance.