CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

Software development is a most concerned field in many of the industries to solve the problems efficiently. However faults occurred at the time of software development would lead to various problems like wrong output generation, failure of software and so on. In this research work, various methods have been proposed in order to predict the faults that are present in the software in various levels.

In the first research work, software fault prediction is done at the design phase to avoid the unnecessary software implementation thus the cost can be saved. The fault prone classes are predicted with the help of UML diagrams using classification approaches. The classification approaches that are applied to predict the fault prone classes are Naïve bias and KNN classification algorithms. The results of this research work proves to provide the better result in terms of avoidance of wrong software implementation and the correction can be made before software implementation.

In the second research work, fault prone class detection is done after software implementation. The relationship between the classes is found in terms of metrics called cohesion and coupling. This is measured by using
the technique called the java reflection which will find the interrelationship between different classes and methods in terms of data communications. Based on this class interrelationship, software faults can be identified well. The result of the proposed research work proves that the software fault can be predicted accurately by analyzing the interrelationship between the methods and classes present in software.

In the third research work, faults that are found in the software are classified to predict the nature and kind of faults. This classification is done by using the methodology called the ANFIS prediction scheme which will classify the faults based on their nature of effect on the software coding. The result of this research work proves that the proposed methodology can find the faults of different kind that are occurred in the software program efficiently. This is done by using the learned knowledge of the faults that are found in the software.

This proposed model could be extended to deal the SDLC phase specific issues rather than just offer one model for different phases in which the roles and responsibilities along with outcomes are entirely different. This modular approach will simplify the monitoring of Software project development stages and assessment against fault at every stage of the development.

7.2 FUTURE WORK

The proposed approach could also be extended for developing project-specific fault prediction, which are more suitable for variety of projects, varying in terms of nature of application, size and related quality factors. These models will improve the applicability, which will enable to offer accurate human judgments in case of software project management. In
addition to that these models will provide greater control over project costs ensure reliable project delivery dates.

The limitation of all the three approaches is that currently they have been tested only with few projects. The validation can be further strengthened by using more number of projects. This work can also be further extended to estimate one of the major quality attributes, the testability, during the analysis phase.

In future, Software fault prediction can be done by using the test cases. The varying test cases can be generated based on the software functioning and their resultant behavior can be analyzed. Relationship between the classes can be identifying by constructing the tree that reflects the semantic relationship between the modules and classes. As a part of future work, Quality Metrics pertaining to the other phases of software engineering like analysis & design can also be developed and their effectiveness in predicting fault is investigated. These metrics derived from the early phases of software life cycle can be of greater utility to the software engineer to assess the high quality of the software being engineered.