CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 CONCLUSION

The objective of the architecture evaluation of the software system is to assure its quality. An exhaustive literature survey was taken up to derive guidelines related to the use of the most suitable method for an architecture assessment process, thereby discovering the similarities and differences between the eight methods. One of the important issues relating to software development is quality, the concept of software architecture (SA) has emerged as the appropriate level for dealing with software quality. This is because the scientific and industrial communities have recognized that SA sets the boundaries for the software qualities of the resulting system. It is recognized that it is not possible to measure the quality attributes of the final system based on SA design alone. This would imply that the detailed design and implementation represent a strict projection of the architecture. The purpose of the evaluation is to analyze the SA to identify potential risks and verify that the quality requirements have been addressed in the design. Other methods include a more abstract evaluation of how the SA fulfills the domain functionality and other nonfunctional qualities.

Software architecture evaluation methods are essential to expose the specific limitations of the architecture. This exercise is necessary to avoid delays in development, deployment, rework, and cost escalation. This work has presented a comparison of the various architecture evaluation methods
and guidelines for selecting the most suitable method for the architecture assessment process. It was observed that ATAM is a robust method for evaluating software architecture. The drawbacks of ATAM such as how to address stakeholder’s diversities, inability to address business strategy viewpoint, non-adaptability to complex systems and the problem in measuring the quality attributes (maintainability, reusability, extensibility, scalability) are addressed to an extent by introducing EATAM, which unfortunately does not satisfy the measure of all the quality attributes. Finally, innovative patterns are introduced. It was proved that applying innovative patterns improves the quality of software architecture to a higher level.

This research adopted a novel approach which was based on the application of innovative patterns, namely, subtraction, multiplication, division, task unification, and attribute dependency change. The subtraction pattern is used to eliminate the redundant and useless components. The multiplication pattern is applied by building a copy of a component with some modification to enhance a specific feature. The division pattern is used to break down complex components into simpler components to better serve the users. The task unification pattern is the way of augmenting usability. During task unification we merge the distinct tasks that can be served by a specific component. Finally the attribute dependency change. In this we add finer granularity to the component by setting a different scope for different category of users.

All these patterns were applied to the case studies and it was established that there was a significant improvement in the design and the quality attributes of the software components. The case study was analyzed and the effect of applying patterns on the quality attributes listed in the table were studied. This suggests that the above patterns are universally applicable.
7.2 **FUTURE ENHANCEMENTS**

There exists a lot of scope for fruitful and productive enhancements and future work. The options for developing a framework which helps to find the suitable patterns to be applied after an accurate evaluation based on formal methods, could be explored. A set of generic templates could be devised so that patterns can be applied objectively across all architecture styles. An accurate mathematical model for representing patterns and their behavior needs to be developed. Integrating patterns into the early phases of software development is also highly desirable. A probabilistic model for suggesting and recommending a suitable pattern could be of great help to software architects. New case tools should be developed for automating this procedure. The representation of patterns could be integrated into UML also.