CHAPTER 7
CONCLUSIONS & FUTURE DIRECTIONS

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Contents</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Conclusions</td>
<td>155</td>
</tr>
<tr>
<td>7.2</td>
<td>Future Directions</td>
<td>156</td>
</tr>
</tbody>
</table>
CHAPTER 7

CONCLUSIONS & FUTURE DIRECTIONS

This chapter provides summary and conclusions of the research work carried out and presented in this thesis. It also provides the future directions of the work that can be carried out further as extensions to this work.

7.1 CONCLUSIONS

In this thesis a methodology for dynamic program invariants to support software evolution has been developed. As part of it initially a technique called Ignoring Unused Variables has been developed which consists of a variable analyzer. It is used to ignore unused variables and consider only used variables and hence infers only relevant invariants. By ignoring irrelevant variables and irrelevant invariants time required to spend by the tools on irrelevant variables and their properties is reduced.

A metrics suite has been proposed for categorizing the relevant variables. This further improves the speed and efficiency of the tools by reducing the effort. These metrics are analytically validated against Weyuker properties and observed that all the applicable properties are fulfilled. The proposed metrics suite also has been empirically validated using two case studies RFV and JLex. Results obtained from these case studies are presented.

Based on the categorization of variables the relevant invariants are also categorized into three types as design invariants, non-design invariants and hybrid invariants. So, whenever a change occurs to a variable first level check should be made to find out whether it is a design variable or non-design variable. If it is a non-design variable and if it is having its impact, to the maximum extent this change will lead to a design equivalent of the software system. At the next level it should be checked that what are the invariants in which this variable is participating. If the change occurs to a design variable then more
care should be taken in implementing this change because it may lead to design change of the software system. It is also to be observed that what are all the invariants that are affected by this change. So, instead of analyzing all the invariants it is sufficient to analyze only those invariants in which the change takes place and other invariants that are affected by this change. Hence, analyzing few among many invariants obviously reduces the time and effort required for analysis and decision making which are very crucial in supporting software evolution.

7.2 FUTURE DIRECTIONS

Whenever a change occurs in an artifact it will be propagated to other related artifacts. From the literature it is evident that invariants play a crucial role in many aspects. Similarly, invariants play an important role in assessing the change. The study regarding impact of change propagation and its maintenance can be explored as a future work.

Usage of invariants in supporting software evolution is proved in this thesis the same can be applied to each and every aspect of evolution like refactoring. The proposed methodology can be applied to all the four types of maintenance; corrective maintenance, adaptive maintenance, perfective maintenance, preventive maintenance and their impact may be studied.

Various types of change propagations have been identified as ripple, wave and avalanche. Ripple type of changes are limited in number and these changes and their propagations are controllable. Wavetype of changes are more in number but still these changes and their propagations can be controlled. Avalanche type of changes are more in number, these changes and their propagations are uncontrollable. Proposed methodology can be applied to study how invariants can be used to control these types of changes and their impact.