ABSTRACT

Evolution happens very often in software systems. The underlying reason for evolution is change in the software. Any change cannot be simply implemented as it comes. A complete study should be made from all perspectives to make a decision regarding implementation of the change. So, behavior of the software has to be analyzed. This can be made either statically or dynamically. Static analysis is analyzing the software without actually executing program. Dynamic analysis is analyzing the software while executing program. Preliminary aspect that influences the decision making is the study about properties that will be affected or disturbed due to the incorporation of change. The other properties or part of the software which will be affected due to propagation of this change also have to be studied. To perform this sort of study, program invariants must be considered. Program invariants are the properties of the software that are to be preserved. Program invariants play a very important role in almost all the phases and aspects of software engineering. They are very much useful in software development. They protect programmers from making errant changes and verify properties of a program. Invariants can be explicitly stated in programs. Programmers can annotate code with invariants but this takes time and effort and also many important invariants will be missed. Invariants can be detected using either static methods or dynamic methods. But, dynamic methods are more effective and extensively used compared to static methods. Diduce, carrot, arnout’s, henkel and diwan, daikon are some of the tools through which invariants can be detected dynamically. Among these tools daikon is the most efficient and extensively used tool. However, currently available dynamic invariant detection tools have some drawbacks which affect their speed and performance. They produce huge list of invariants which include irrelevant invariants also. Analyzing the huge list of invariants require more time and effort which are of waste and also affects cost, leads to wrong decisions which causes huge loss and damage in many cases. To avoid these adverse effects it is utmost required to have a methodology to support software evolution.

As a solution to above problems, this research work aims at ignoring unused variables initially. So that inferred invariants will be only relevant invariants. These will be less in number drastically when compared to present huge list of invariants which include irrelevant invariants also. The irrelevant invariants are due to unused variables. Ignoring unused variables reduces the time and effort required to analyze the invariants and inturn cost as well it will not lead to wrong
decisions which may cause loss or damage. To further improve all these factors, the following aspects are considered in this thesis. Even all the relevant invariants need not be analyzed when a change occurs. It is sufficient to analyze or investigate only those invariants which will be involved or affected by the change. As the change occurs to variables at the basic level, it is very important to check that whether this change will lead to major or minor change in the software. It is required to check whether this change leads to a design change in the software or equivalent design of the software. For this purpose, variables have to be categorized as design variables and non-design variables. To do so, this research work intends to provide a scientific basis with the proposal of a metrics suite. The proposed metrics suite has been validated both analytically as well as empirically.

With the categorization of variables, invariants can also be categorized into various types as design invariants, non-design invariants and hybrid invariants. So, in this manner it is very easy to trace the change and its propagation. Hence, it requires very less time and effort to analyze invariants with respect to decision making and change implementation. If the change is in design invariants more care is required and decisions have to be made with utmost care because they may lead to design changes. A solution for all above discussed aspects is the methodology for dynamic program invariants to support software evolution. It helps in reducing time, effort and cost; and also aids in decisions making and change implementation. So that no errant changes occur that will cause any type of loss or damage to the software systems and also to the scenarios in which they are used. Entire methodology has been applied over two case studies namely RFV and JLex, two open source software. Satisfactory results have been obtained and are presented in the thesis.

**********