Currently, the cost of development and maintenance of software systems has become unacceptably high and maintenance costs outweigh the development costs of a typical large software system over its life cycle. Also, if time period increases, then maintenance costs for a typical organization takes a greater share of the total computing budget. Thus, a prime concern of software scientists is to develop a good quality software with minimum cost and within the limited allowed time period. For this, the basic requirement is the development of programs which are easy to understand and maintain.

Due to increasing complexity and costs of software development and maintenance, the need for objective measurement of software systems has increased. From time to time, different researchers have attempted to define and measure the "complexity" of a computer program. Software complexity directly affects maintenance activities like software understandability, modifiability and software testability. Predicting software complexity can save millions in maintenance. Clearly, if complexities could somehow be identified, then development, maintenance and testing procedures can be adjusted accordingly. This concern has led to various studies directed towards identification of factors contributing to complexity of programs and development of complexity-reducing methodologies.

In software industry, a great deal of effort is now being devoted for searching ways to produce reliable software and to reduce the cost of maintenance. Ways to predict complexity and high maintenance costs should originate in software design. Earlier we isolate a fault in software development life cycle, the lesser will be the cost to rectify it. Software development life cycle can be monitored and controlled effectively through the application of measurement theory. Basically, measurement is the quantification of the observed behavioural properties of objects and the systems, and their behaviour is predicted from these quantified properties. As a result, the increasing importance is being attached to the measurement of basic characteristics of software systems such as size, effort, complexity, productivity, quality, presence of faults, testability and so on. The predication of several such aspects of software systems is the motivation behind the study and design of software metrics.

Software metrics refer to a broad range of measures of various characteristics of software systems. These can serve as measures of software products for the purpose of comparison, cost estimation, fault prediction and forecasting. Cook defined a software metric as a "Measure of the extent or degree to which a product possesses and exhibits a certain quality, property or attribute". Software metrics provide a quantitative view of various aspects of software systems and can be effectively utilized in planning, controlling, and monitoring the software development process and software product. They may be invaluable to software managers in allocating resources necessary for
development, maintenance, quality control, and testing activities. For example, by knowing the complexity and testability of a program, we may utilize testing resources more effectively because a program, with more complexity and low testability requires more testing than a program with less complexity and high testability. They may, also, help in predicting about its quality as well.

Thus, the software metrics have great importance for software systems. These are useful for the study, analysis, prediction and minimization of development and maintenance costs of software systems. We have investigated the software complexity metrics following the concepts of software science and control complexity for four modern programming languages, namely, Pascal, C Modula-2 and Ada. While Pascal and C languages have been used in earlier studies also, there are few studies in which Ada programs have been studied. But, as far as we know, there are hardly any studies in which software metrics have been applied to Modula-2.

We have proposed new software complexity measures based on various aspects of complexity of a software module. We have calculated values of various complexity measures for the above mentioned programming languages by applying three term length estimator. We have compared these values with the other existing length estimators and found a close relationship between the two. We have also suggested some modifications to the counting rules of programming languages and thus resolved a contradiction about the level of the Ada language. By using the process of sensitivity analysis, we have proposed a method for measuring the testability (amount of testing required to reveal expected faults) of a computer program. By following Lipow's ideas, fault rates (number of faults per line of code) for Pascal, C, and FORTRAN languages have also been calculated. We, also, have investigated a method through which we can predict the number of faults in a computer program. We have conducted a class-room experiment to apply the fault prediction method to computer programs and to find out the effect of program complexity on predicted faults. We have tried to establish the relationships among software complexity metrics, faults and testability and the results were found to agree with intuition.

Thesis Organization

The thesis has been organized into nine chapters:

- First chapter covers the introductory part of this study and the motivation behind it.
- Second chapter includes the survey and description of various software metrics.
- Third chapter describes the proposed three term program length estimator, its experimental study and its comparative analysis with other program length estimators. It explains the variations in counting rules/strategies used and their effect on solving the contradiction raised
earlier about the level of Ada language. An attempt has, also, been made for predicting size and complexity of a software module before coding.

Chapter four discusses the design of a new program weighted complexity measure. The proposed metric quantifies four prominent complexity factors, namely, size of a software module, types of control structures, their nesting level, and position of a statement in the logic of a program. An attempt has, also, been made to resolve some existing controversial issues of McCabe's metric.

Fifth chapter reports the relationships among the software complexity metrics. It, also, includes the criterion which may be used for the categorization of software systems as "decision bound" and "computation bound" systems.

Chapter six explains the connection between program complexity and faults predicted and it supports the intuition that more the complexity \(\rightarrow\) more the predicted number of faults. For faults prediction, fault rates for C, Pascal and FORTRAN languages have, also, been reported.

Seventh chapter describes the concept of testability and the conditions for software failure. A method for calculating testability from sensitivity analysis has been explained. It, also, establishes a relationship between program complexity and testability and the result is in agreement with intuition that more the complexity, more the amount of testing required to reveal expected faults from it.

Chapter eight explains the bebugging technique for faults prediction and its experimental study through a designed class-room experiment to know the effect of complexity over the number of revealed faults. It gives an experimental support to the intuition that the number of faults depends directly on complexity. Comparison between experimentally predicted faults (through bebugging method) and theoretically predicted faults (Lipow's method) has also been done and found a positive correlation between the two. However, experimentally predicted results are more encouraging than Lipow's method. The reason for this may be that experimental results are based on realistic data while Lipow's method is based on theoretical assumptions which may have some problems.

Ninth chapter includes a discussion of the conclusions that can be drawn from this study and the limitations of the present work and also suggests further possibility of research in this area.