AN IMPLEMENTATION OF THE SCHICK-WOLVERTON AND THE JELINSKI-MORANDA SOFTWARE RELIABILITY MODELS

by

JOHNNIE OTIS RANKIN

B.S., Oklahoma State University, 1970

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements of the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1982

Approved by:

Major Professor

ACKNOWLEDGEMENTS

I would like to thank several people for the invaluable assistance given me during my efforts with this project. The first is my major advisor, Dr. David A. Gustafson. Dr. Gustafson's direction and personal interest in my project were what kept me going through difficult times. Additionally, Mr. Robert Young and Mr. Carlos Qualls provided me with expert technical assistance during the course of my work. And finally, I would like to thank Mrs. Mary Beth Cole for her tremendous support. Without all of these people and their contributions, I could not have completed this work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE

INTRODUCTION

One of the major tasks facing the software engineer in the program development cycle is the determination of when a sufficient amount of testing has been performed. This is not an easy decision to make. Determining that a sufficient amount of testing has been conducted is in a sense a statement of the degree of the correctness of a software package. The degree of difficulty which is associated with this decision drastically increases as the size of the software package increases. In addition, there are factors which tend to exert pressure to reach this determination as expeditiously as possible. These are mostly economic factors of course, such as, time, the cost of large manpower requirements to perform the testing functions, the cost of the associated administrative requirements to perform the testing function, and the overall cost of the testing effort. It is widely accepted that the testing and certification effort is one of the more expensive portions of the developmental cycle and thus any efforts toward allowing this determination to be made accurately and as soon as possible would be welcome by the software engineer.

The Software Reliability Model has evolved as an aid in making this determination[3, 9, 10]. The process of

determining sufficiency of testing can be accomplished with more confidence, less time, and reduced cost by the use of a valid error prediction model [12,13,15]. Thus, the Software Reliability Model has become an important factor in the area of testing.

It should not be construed that this model is the answer to each and every problem in testing. It is obviously not. The Software Reliability Model would be utilized most effectively in an integrated role with other alternatives to determining sufficiency of the testing effort. These other alternatives include number of errors discovered over a period of time, the number of paths of a program executed during testing and this number's relationship with the total paths of the program [3,5,13], and the criticality of those errors discovered. Although these alternatives are frequently used in the decision making process to determine this sufficiency of testing [13], there is certainly room for improvement in the area. The Software Reliability Model would be useful as one of the collection of indicators that the software engineer may use to reach a sound decision as to when enough testing has been done.

With this statement of the importance of testing and specifically the determination of when enough of the testing process has been performed, the purpose of this project is twofold. The first is to provide the Department of Computer Science at Kansas State University with a tool that could be used in classroom applications. Basic software engineering

courses offered in the department include material discussing the program development cycle quite extensively, and naturally enough the testing phase of the cycle is also extensively covered. It is intended this implementation be integrated into the testing material of the development cycle as an indication of how the Software Reliability Model could be integrated into the decision making process to determine sufficiency of testing.

The second purpose of this project is to increase my expertise and exposure in an area I will have continued exposure to within the military environment, that of testing, and to provide me with the framework of a package that can be utilized at other computer facilities performing various functions within the Army.

This implementation is an interactive program which is designed to compute the estimated reliability associated with a number of errors in a partially debugged sofware package. The implementation offers the use of two currently accepted Software Reliability Models, the Schick-Wolverton and the Jelinski-Moranda model. Appendices E and F have additional details of these two models. The implementation has been designed to support a user friendly approach to the interactive process by providing two distinct levels of interaction, an expanded instruction sequence for a user unfamiliar with the program and a minimum instruction sequence for a user who is more experienced with the program execution sequence. See Appendix I for details of the degree of interaction and instruction sequences offered.

Besides reliability estimates, the implementation incorporates other meaningful estimates of importance to the managerial level and to the software engineer. These estimates are the mean time to failure of the software project and the time to discover all remaining errors within the software project. As a byproduct of the reliability models in general, these two estimates have an important place in the determination of sufficiency of testing by the very nature of the information they convey. An estimate which is an accurate reflection of a mean time to failure rate would be of keen interest to a manager of a project. The same type of generalized statement can also be made about the time to discover all errors within the software package estimate. This information could prove to be invaluable in projecting schedules, curtailing costs, determining sufficiency of testing, or any number of other decisions that the manager and/or software engineer must make during the development cycle of a software project.

All computations of this implementation may be presented on two solution forms. These are a tabular form on which the reliability estimate, mean time to failure, time to discover all errors within the package, and other statistical information is presented, and a graphical form on which reliability is plotted versus mean time to failure for the software package. See Appendix H for examples of these solution forms.

CHAPTER TWO
DESIGN ISSUES

In the preliminary design phase of this implementation, the overall effort was directed to designing an implementation package that would meet the purposes, as outlined in Chapter 1, for doing this project. Two basic goals surfaced from this effort. They were first of all to design a system implementation package that was highly "user friendly" in nature and secondly to design a system implementation package as far as future modifications were concerned and thus increase the usefulness of the implementation to the Department of Computer Science. See Appendix D for a detailed discussion of modification issues relative to this project.

In deciding how to approach the design of the overall project, it became obvious that the package should be highly modular in nature, with functions of querying the user to collect data necessary for the models to operate on and the actual computations themselves being performed within separate packages. The mechanics of drawing the solutions would also be modularized into separate packages. This approach led to the development of three separate programs, one to interface with the user and collect all necessary data to perform the computations on, and one for each display device used. The initial planning called for two devices to be incorporated, the Chromatics display device and the Plotter

display device. This decision avoided one monstrously large implementation package and allowed the development of three medium sized packages.

A subsequent issue addressed in the design phase was the form of the solution offered to the user. Although the initial planning was for a graphical solution, it became obvious in investigating and researching the models to be used, that they offered information of importance that would not be displayable on a graph. This led to the decision to incorporate a tabular type solution format and offer the user the choice of which, or both, format he desired. This decision turned out to be wise for it allowed a more accurate presentation of the computation of the model than can be interpolated from a graph. However, I did not feel that this increase in detail completely negated the value of a graphical solution in that the graphical solution is extremely valuable in showing trends in the data collected during the testing cycle of some software project.

The issue of providing a relatively user friendly interface was easily solved. Two levels of interaction were chosen, as detailed in Chapter 1 and Appendix I, and the decision was made to provide a "help" function to assist the user in moments of indecision as to the proper response to a program generated query.

In retrospect, I am firmly convinced that this decision was the proper and correct one. I feel I have accomplished

a design that will facilitate the incorporation of modifications easily and efficiently, and am certain that my design assisted greatly in the programming and debugging phases of the implementation.

CHAPTER THREE
TESTING AND VALIDATION

Testing of this project was difficult at best. In the testing process, three phases were used. These were exercising the user interface, verifying the accuracy of the implementations of the models, and verifying the accuracy of the graphical solutions. As in most projects, more time was spent in debugging and testing than in the actual programming. The approach for each phase is presented below.

To thoroughly test the interface of the implementation with the user, two steps were used. The first was to exercise each decision node of the interface procedures and the second was to utilize 24 undergraduate students to separately execute the program and offer a critique of its interface potential. I found this latter step to be of immeasurable value. Through the candid remarks of these student testers, I was able to refine initial instructions, queries, and assistance messages to the user to provide a meaningful, straight forward series of directions. This obviously enhances the ability for someone unfamiliar with a statistical reliability package to be able to successfully execute this implementation. No formal data as to the number of errors discovered during step 1 of this phase or concerning those suggestions made during step 2 of this phase was kept.

In exercising each decision node of the interface procedures with the user, errors were discovered and corrected. As each of these decision nodes in the interface itself is dependent upon a user input, this step was actually easy to accomplish.

Verifying the accuracy of the program computations was inherently more complicated than the exercise of the user interface. This is obviously a function of the highly mathematical nature of the models used. The accuracy was verified by hand calculating the various forms of solutions of each model over a range of inputs. See Appendix G for a representative sample of inputs used in this verification of model accuracy. Some difficulty was encountered in choosing the inputs to examine because the size and number of the input parameters that could be successfully calculated by hand was limited. Nevertheless, the results of these efforts indicate a sound basis for judging the implementation calculations to be correct.

Finally, the last phase of the testing process was easily accomplished after the computations of the program were verified. This last phase was the verification of the graphical data and that was of course very dependent upon the model calculations being correct. Once this fact was established, this phase became an exercise in verification of the conversion of the model data to x and y graphical coordinates. Numerous graphs were analyzed and the results were positive in that the graphs are accurate.

CHAPTER FOUR

CONCLUSIONS

I feel that this project has satisfied my purposes for doing it. I have certainly increased my expertise in the area of software reliability and the area of reliability in general.

The implementation works well but after being so actively involved, I can see a necessary addition to make this implementation particularly useful to the Department of Computer Science. This addition would be in the area of adding display devices used.

I feel that at least two more types of display devices could be added with relatively little difficulty. The two types I would recommend are the Spinwriter device and the CRT terminal itself. The addition of the Spinwriter device would provide an additional hardcopy capability to the implementation. At present, only the plotter offers the hardcopy capability. The addition of the CRT as a display device would provide the implementation with an increased flexibility, if only in so much as increasing the number of devices available for use. To be widely used in the Department, more devices will be needed and the CRT addition could certainly do that. I have expanded on the issue of modifications to this implementation in Appendix D.

Finally, in evaluating this implementation, I must also consider the negative aspects. I feel there was one major detriment to my efforts and that was the language chosen to do this implementation, PASCAL. PASCAL was not the language suited for this application. FORTRAN is much better suited due to its power in handling the arithmetic computations which were necessary to perform. The very nature of the computational models used in this implementation is highly mathematical. The absence of capabilities in the Interdata implementation of PASCAL such as mixed mode arithmetic, exponentiation, the standard functions used to raise the natural logarithm base to a power, a square root capability, and the inability to directly write real numbers caused me many hours of grief. A typical implementation of the FORTRAN language would have solved these problems, albeit creating some self-documentation short comings in the process.

APPENDIX A

LOGIC AND DATA FLOW OF THE IMPLEMENTATION

The logic flow of the main program of this implementation is divided into five phases. These are the establishment of a prompting level, querying and obtaining user data, performing the computations upon the data by the applicable reliability model, preparing the user selected solution forms for display, and presenting the selected solution forms to the user. This flow is represented in the Figure 1. Also Appendices B and C for other pertinent information concerning span of control during execution of this implementation and specifications and descriptions of modules of the main program and the display driver programs.

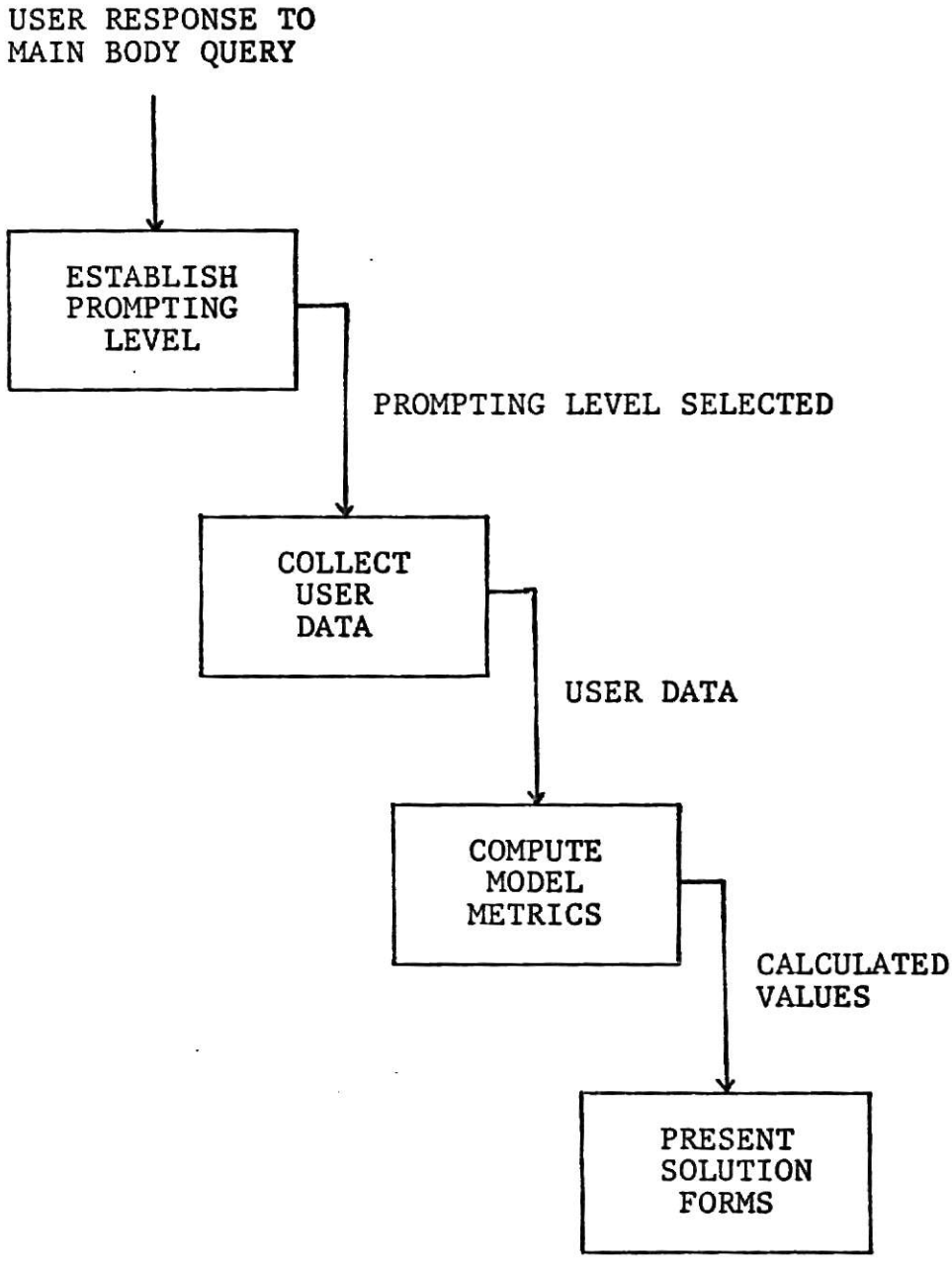

Figure 1. General Logic Flow of the Main Program.

The logic flow of the main program of the implementation is summarized as follows, by procedure function.

a. The user is queried as to which level of interactive prompting he desires. There are two options which are available, full or partial prompting. This initial interactive session is conducted through the main body of the program. Once the user has decided upon the level of prompting option procedure where the remainder of the execution is controlled.

b. The prompting procedure selected again controls the execution sequence of the main program. Specific functions performed are as follows:

1. Directs the user through the inputs which are necessary to collect all data required to compute the estimates of solutions by the respective reliability models. These are as follows:

a. Which reliability model to use.

b. Which solution form is desired.

c. The scale of the mean time to failure axis if the graphical solution was selected.

d. The number of errors estimated to be initially present in the software package.

e. The number of error testing intervals.

f. The time length associated with each of the error testing intervals.

2. After successfully collecting all input data, the prompting procedure invokes the procedures to compute

the solution forms of the program. There are separate procedures for the Schick-Wolverton and the Jelinski-Moranda models.

3. After computation of the model results, the prompting procedure invokes the respective procedures to load the results computed into the form which was selected by the user. Again, this form may be graphical or tabular. There are separate procedures to load Schick-Wolverton and Jelinski-Moranda data into the tabular solution form and to convert the data into coordinates to be graphed.

4. If the tabular solution form was selected, the prompting procedure invokes the tabular solution printing procedure. If both solution forms were selected, the tabular solution form is presented to the user first.

5. If the graphical solution was selected, the prompting procedure invokes a procedure used to obtain the display device the user desires to use. The candidates are the Chromatics or Plotter display devices.

6. After the device has been specified, the prompting procedure invokes a procedure which in turn invokes the particular display device driver program selected by the user.

7. Upon finishing the above tasks, the prompting procedure passes control back to the main body of the program.

c. Upon receipt of control from the prompting procedure, the implementation program is terminated.

d. At each step of the prompting procedure, interaction is carried on with the user in a "user friendly" fashion. All input data is for legality checked and the user notified and asked to reenter that data found to be in error.

e. Finally, at each step of the prompting procedure, access to an assistance procedure is provided for the convenience of the user.

The logical flow of the display driver program portion of this implementation is divided into three phases. These phases are receiving control from the main program, drawing and labeling the graphical framework, and then drawing the graphical solutions themselves. See Appendices B and C for other pertinent information concerning logic flow, specifications and descriptions of modules of the display driver programs. Flow within the display driver program is represented in Figure 2.

The logic flow of the display driver programs is summarized as follows:

a. Receipt of control and the graphical coordinates to be plotted is received by the main body of the driver program. The procedure to control the activities of the driver program is then invoked.

b. The controller procedure directs all further execution of the program. This procedure performs the following functions:

1. Invokes the system initialization procedure to draw and label the graphical framework.

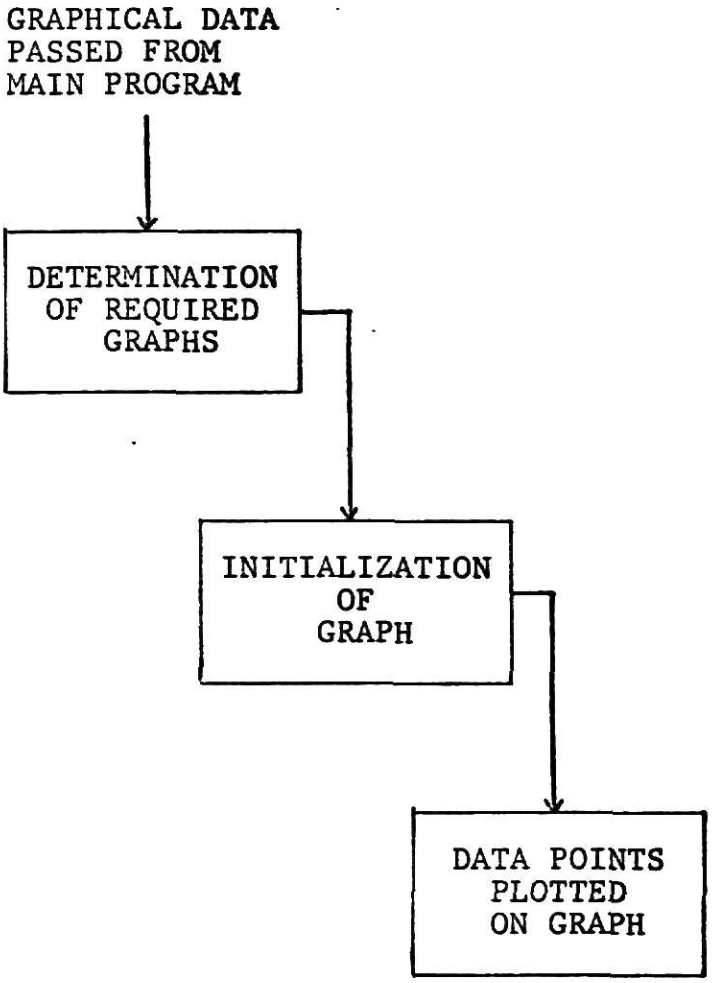

Figure 2. General Logic Flow of Display Driver Program.

2. Invokes the graph drawing procedure to plot the coordinates on the graphical framework.

3. Returns control to the main body of the driver program.

c. The procedure used to initialize the system uses the primitive commands of the respective display device to draw the graphical framework, to label the intervals of the x and y axis, to label each axis, and to provide the legend to enable the user to distinguish data presented on the graph.

d. The procedure used to actually draw the graphs again interfaces with the primitives of the associated display device to perform the task of drawing lines at the proper locations.

APPENDIX B

IMPLEMENTATION STRUCTURE AND SPAN OF CONTROL

The program structure and span of control of certain procedures are depicted in the Figures 3 through 9. Control is indicated in each figure by the connecting line. The General Implementation Schematic (Figure 3) is successively broken down to provide detail as the logic and control flow.

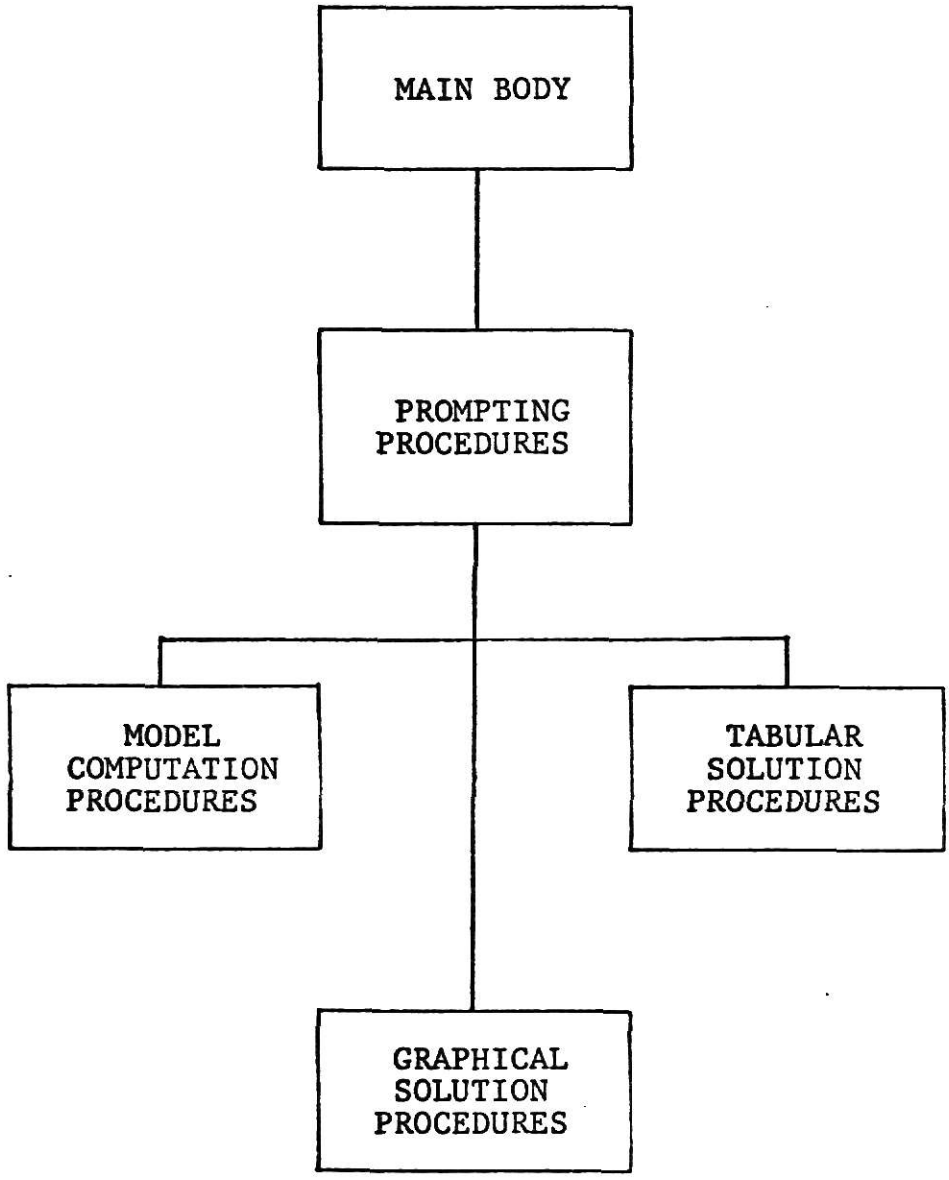

Figure 3. General Implementation Schematic.

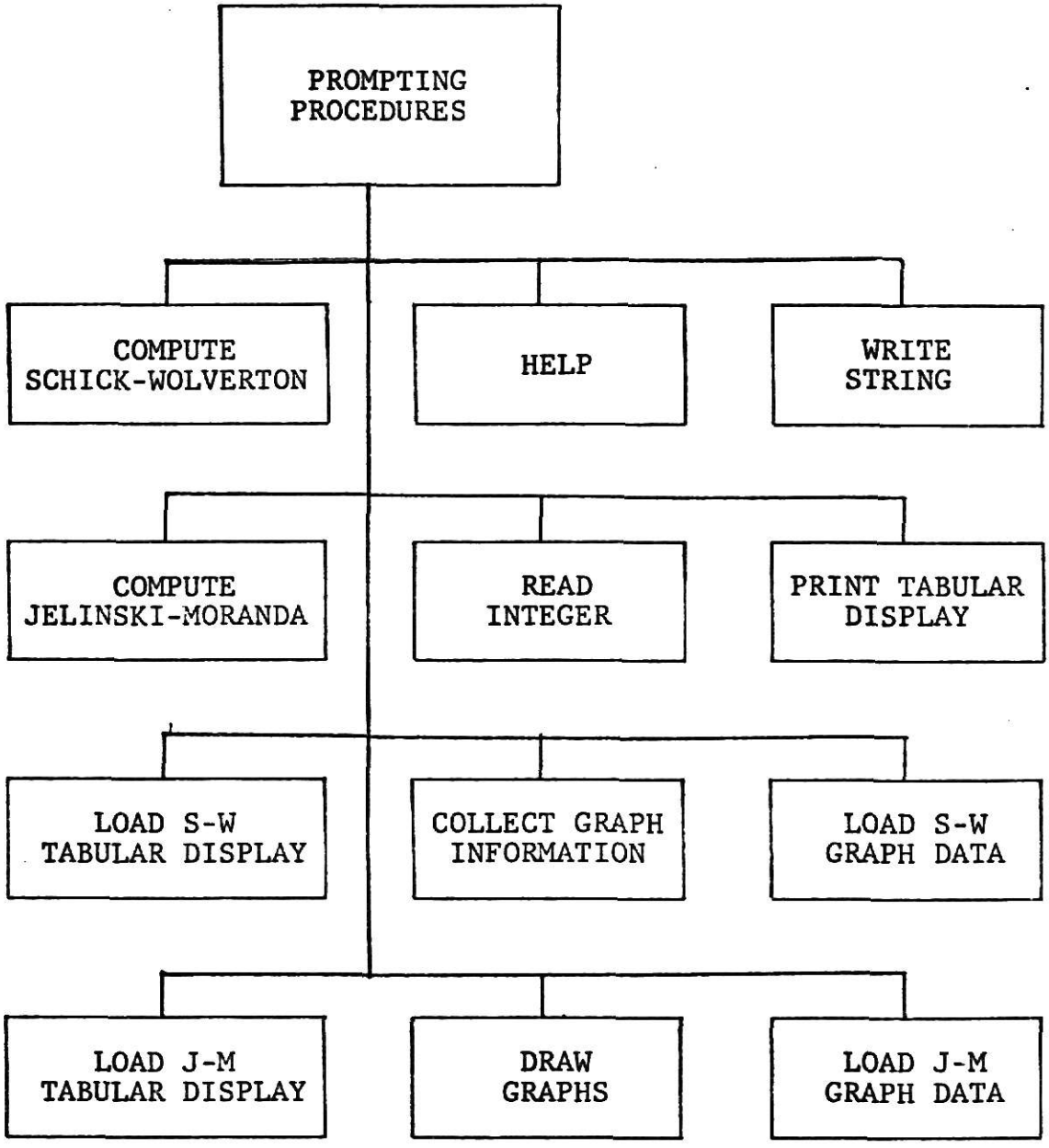

Figure 4. Prompting Procedures Schematic.

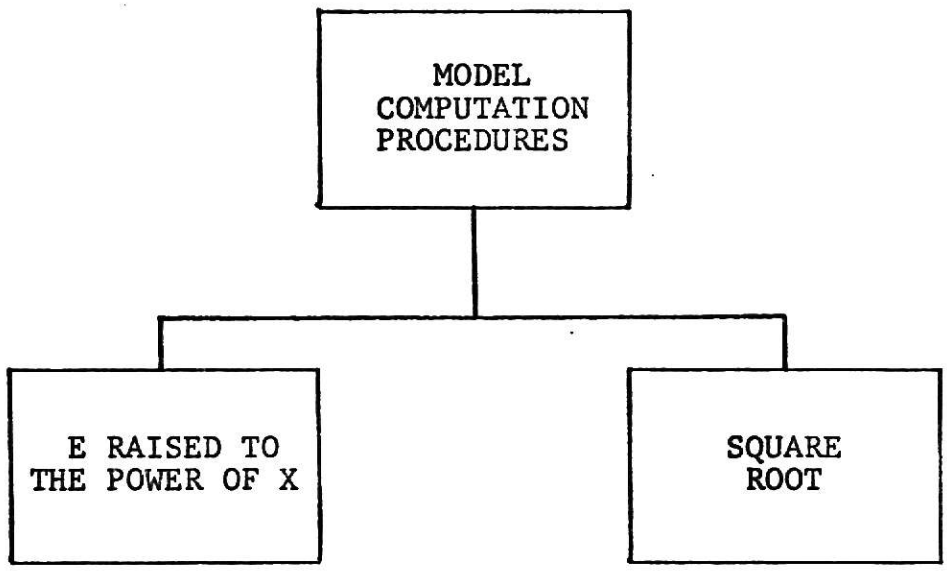

Figure 5. Model Computation Schematic.

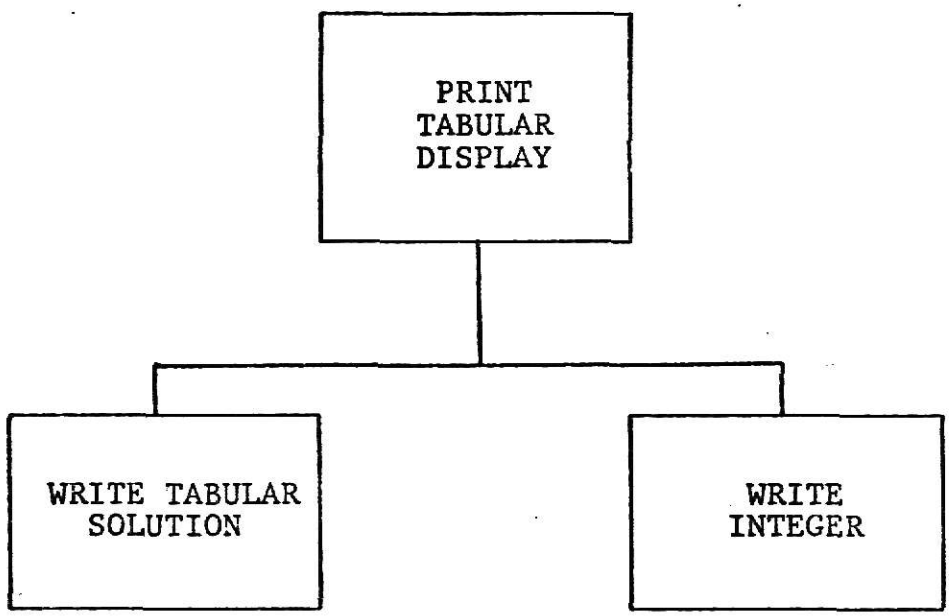

Figure 6. Print Tabular Display Schematic.

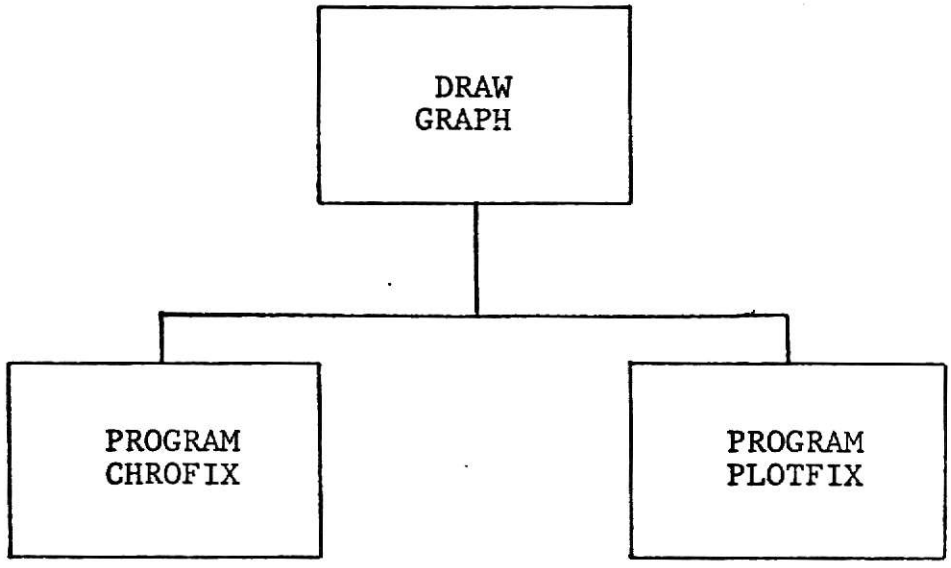

Figure 7. Graphical Solution Schematic.

NOTE: Invocations of Program Chrofix and Program Plotfix are external to the main program.

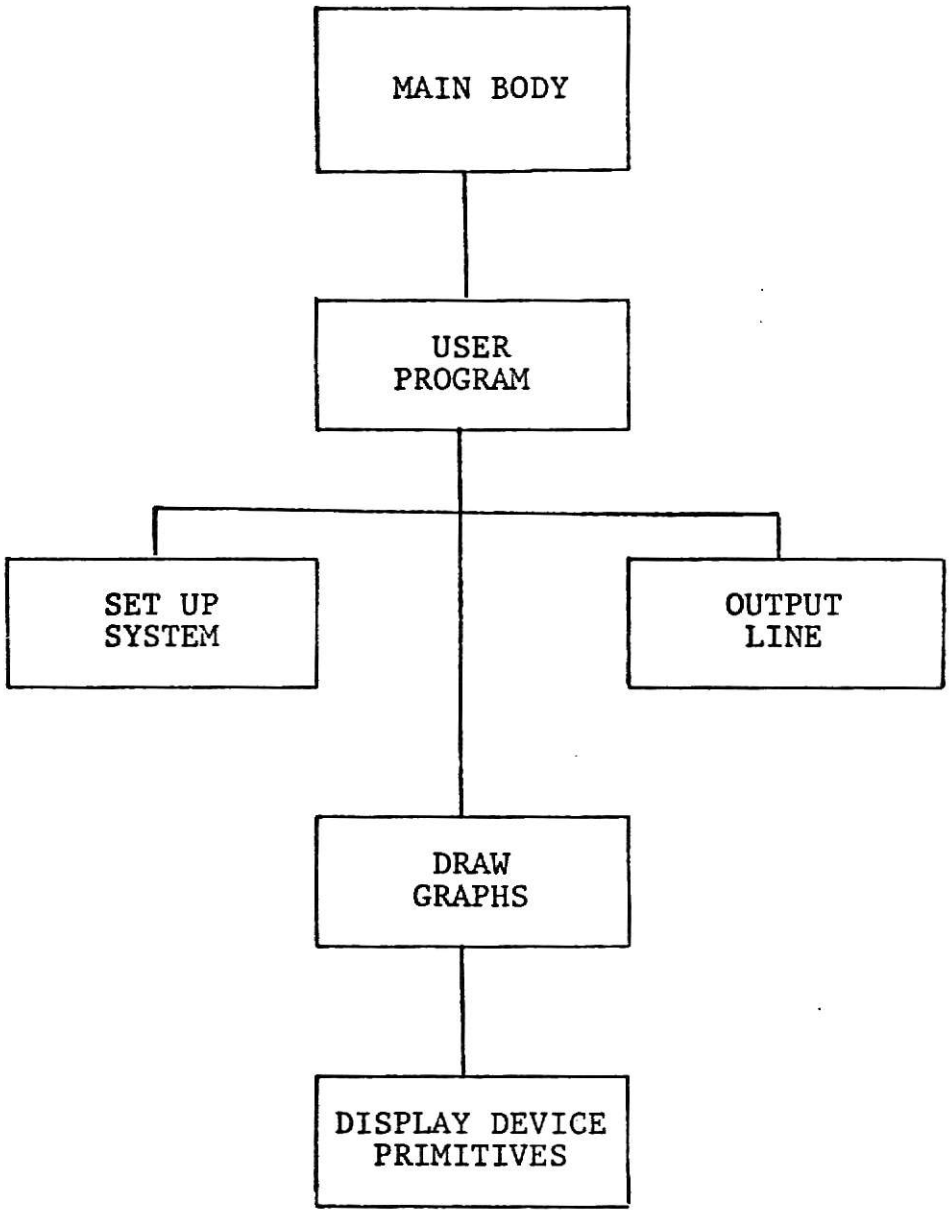

Figure 8. Chromatics Display Device Schematic.

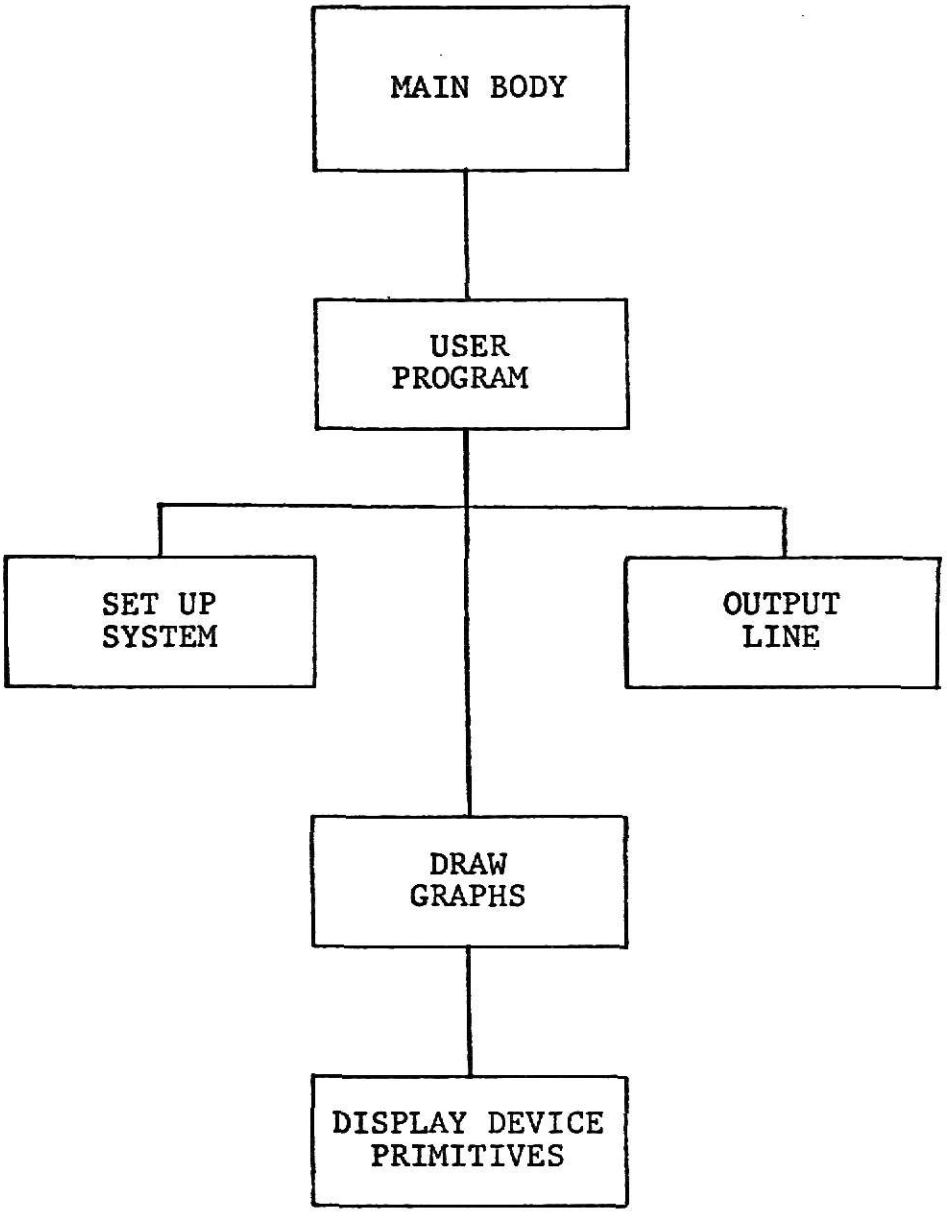

Figure 9. Plotter Display Device Schematic.

APPENDIX C

MODULE SPECIFICATIONS/DESCRIPTIONS

As each major function of the implementation is accomplished through a specific procedure or procedures, the formal specifications and descriptions of the procedures are described below. Additional details of each procedure are provided in Appendix I.

Procedures of the main program are as follows:

a. Procedure Provide Full Prompting. This procedure provides an elaborate interface with the user to collect the information necessary to perform the model calculations and present the solutions. Questions are preceded with complete explanations of what the question is and what the possible answers are. Access to an assistance procedure is provided with each question. The procedure controls the execution sequence of the main program. This execution sequence and the span of control of this procedure is detailed in Appendices A and B, respectively.

This procedure is invoked from the main body with no parameters. The procedure exercises access to global data variables used to indicate the following:

1. The number of errors initially present in the software package being analyzed.