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.

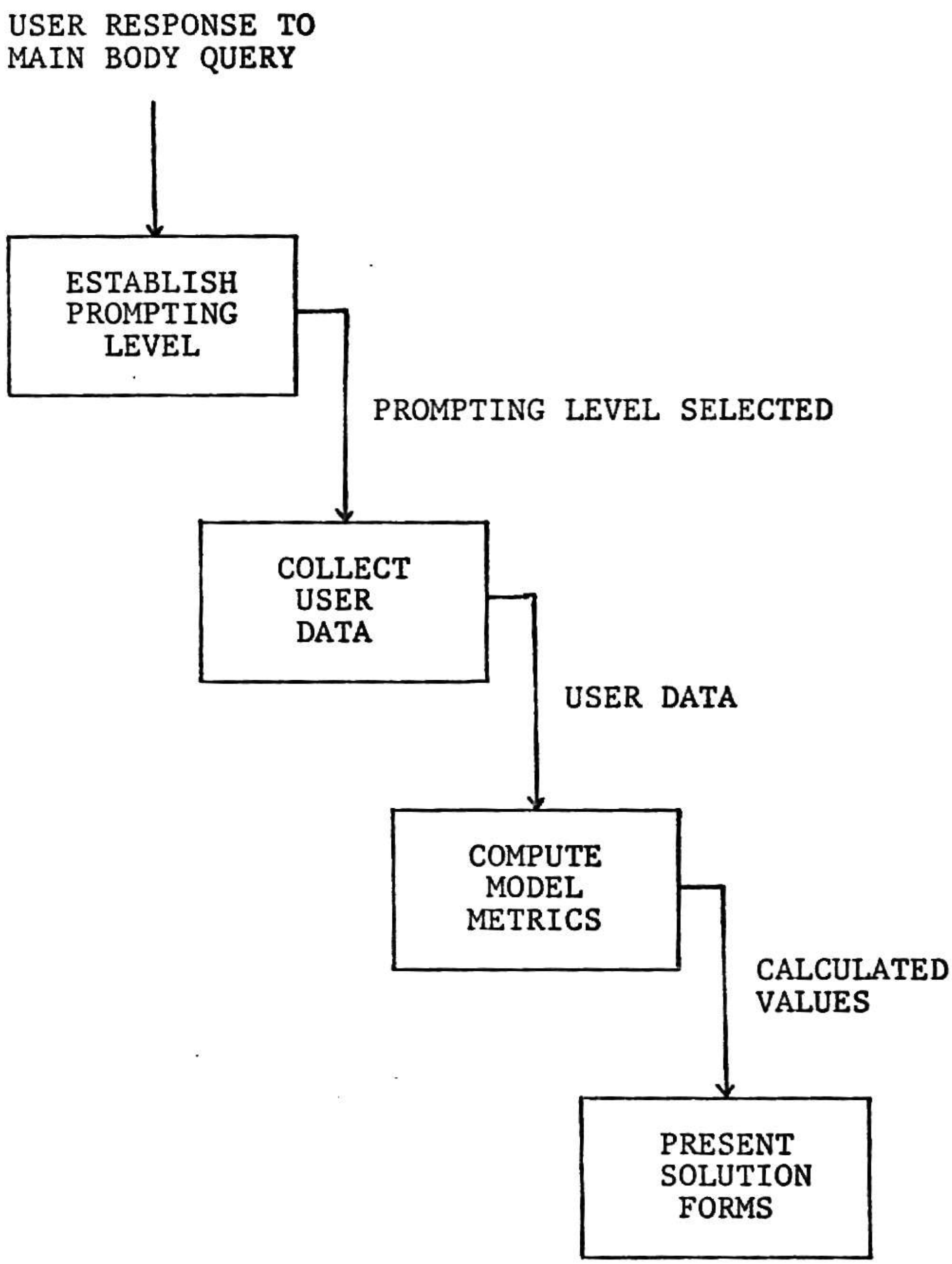

Figure 1. General Logic Flow of the Main Program.

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.

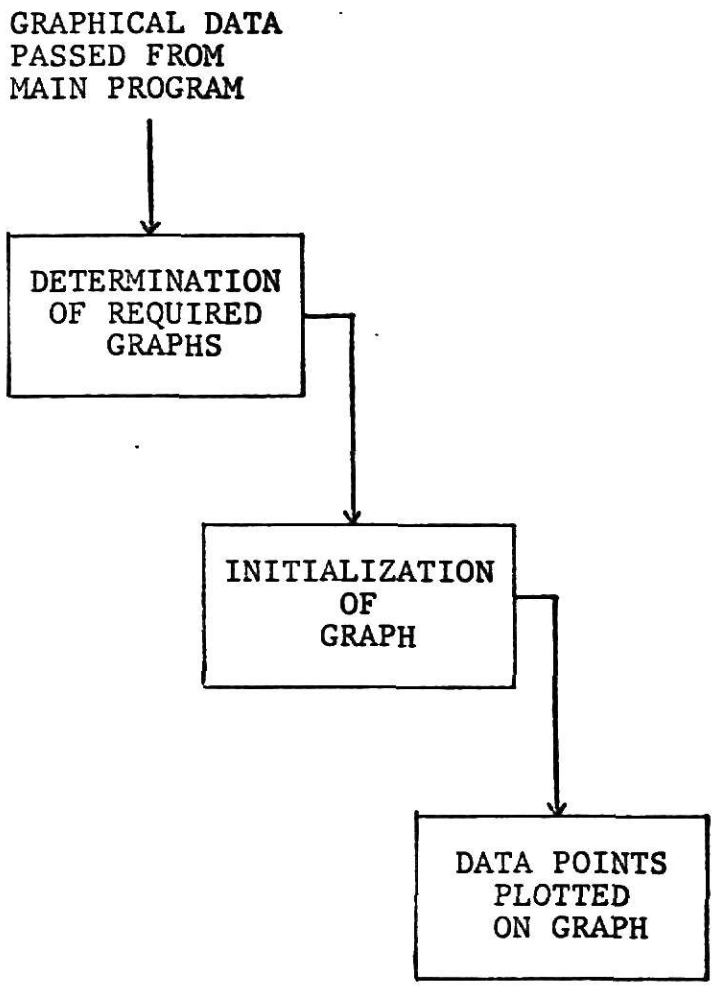

Figure 2. General Logic Flow of Display Driver Program.

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.

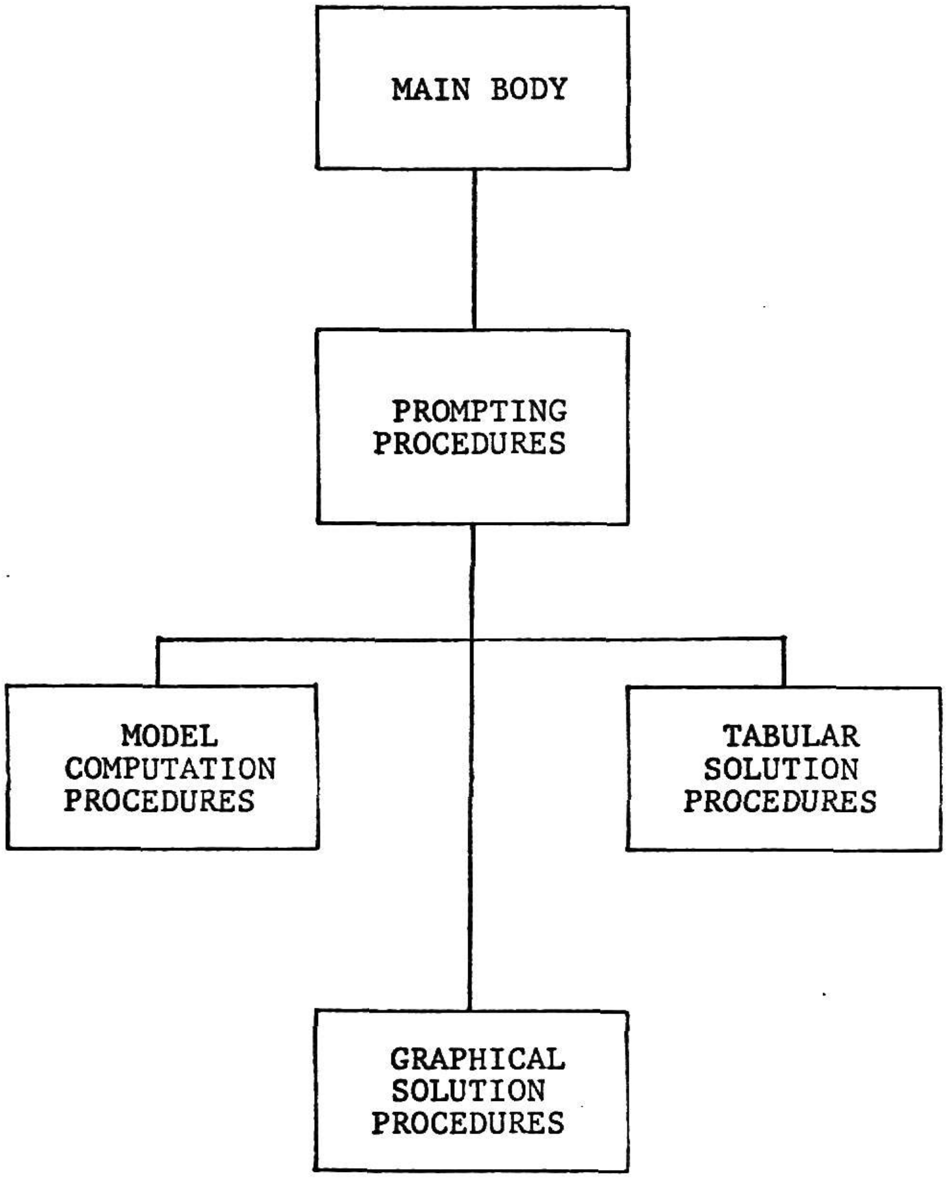

Figure 3. General Implementation Schematic.

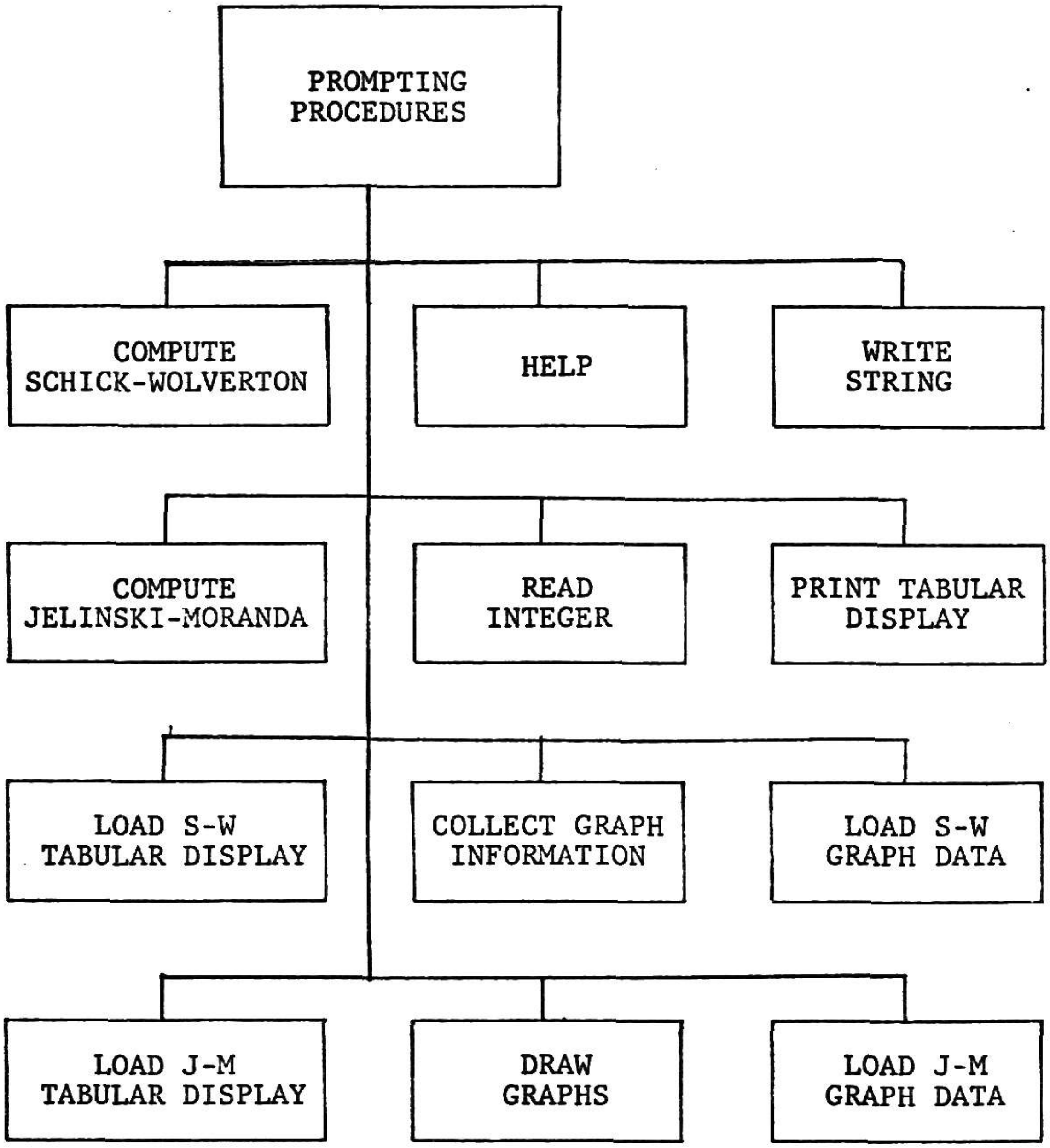

Figure 4. Prompting Procedures Schematic.

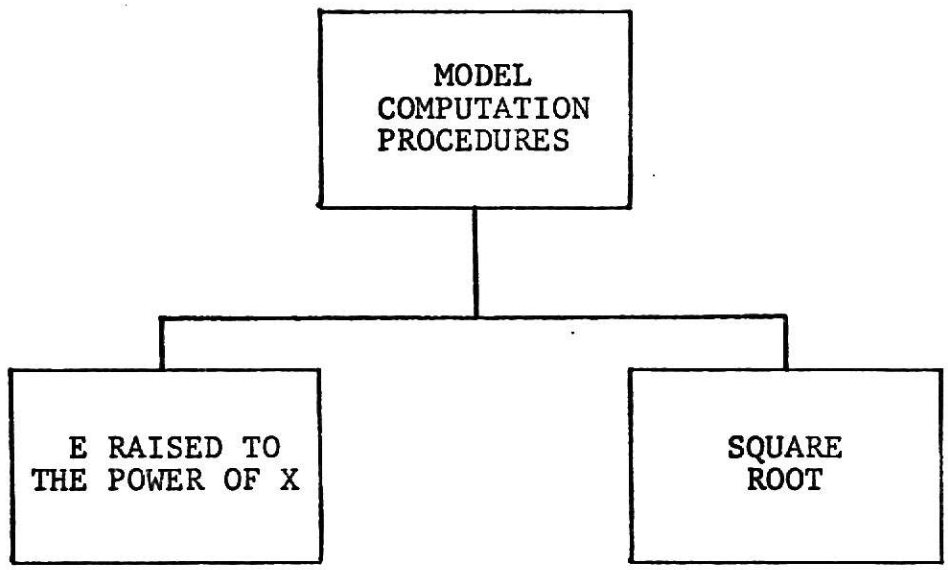

Figure 5. Model Computation Schematic.

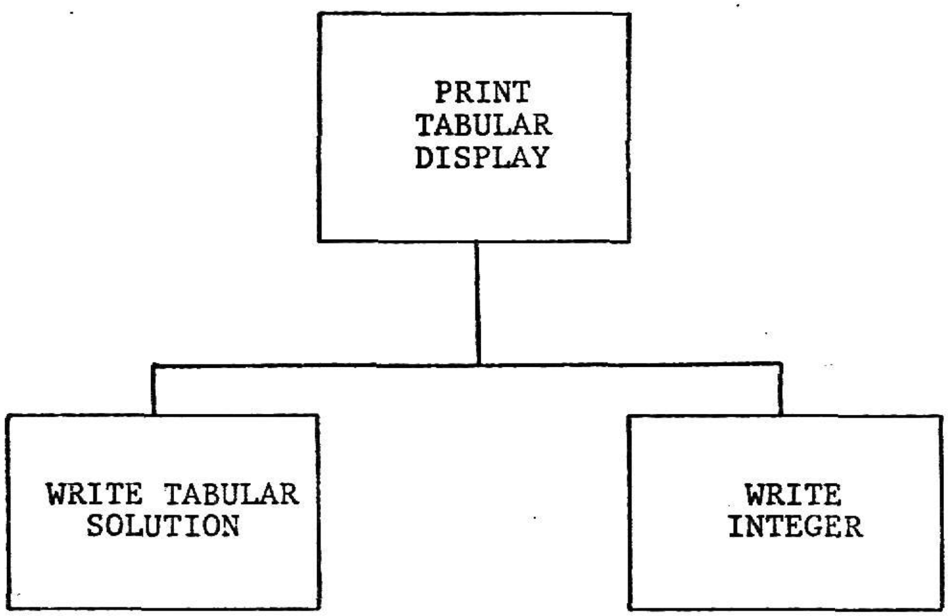

Figure 6. Print Tabular Display Schematic.

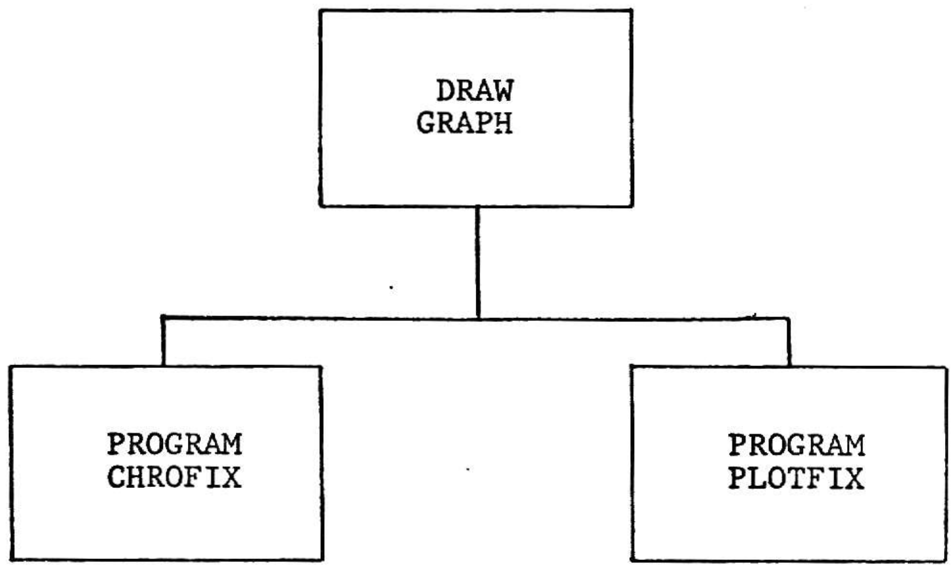

Figure 7. Graphical Solution Schematic.

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

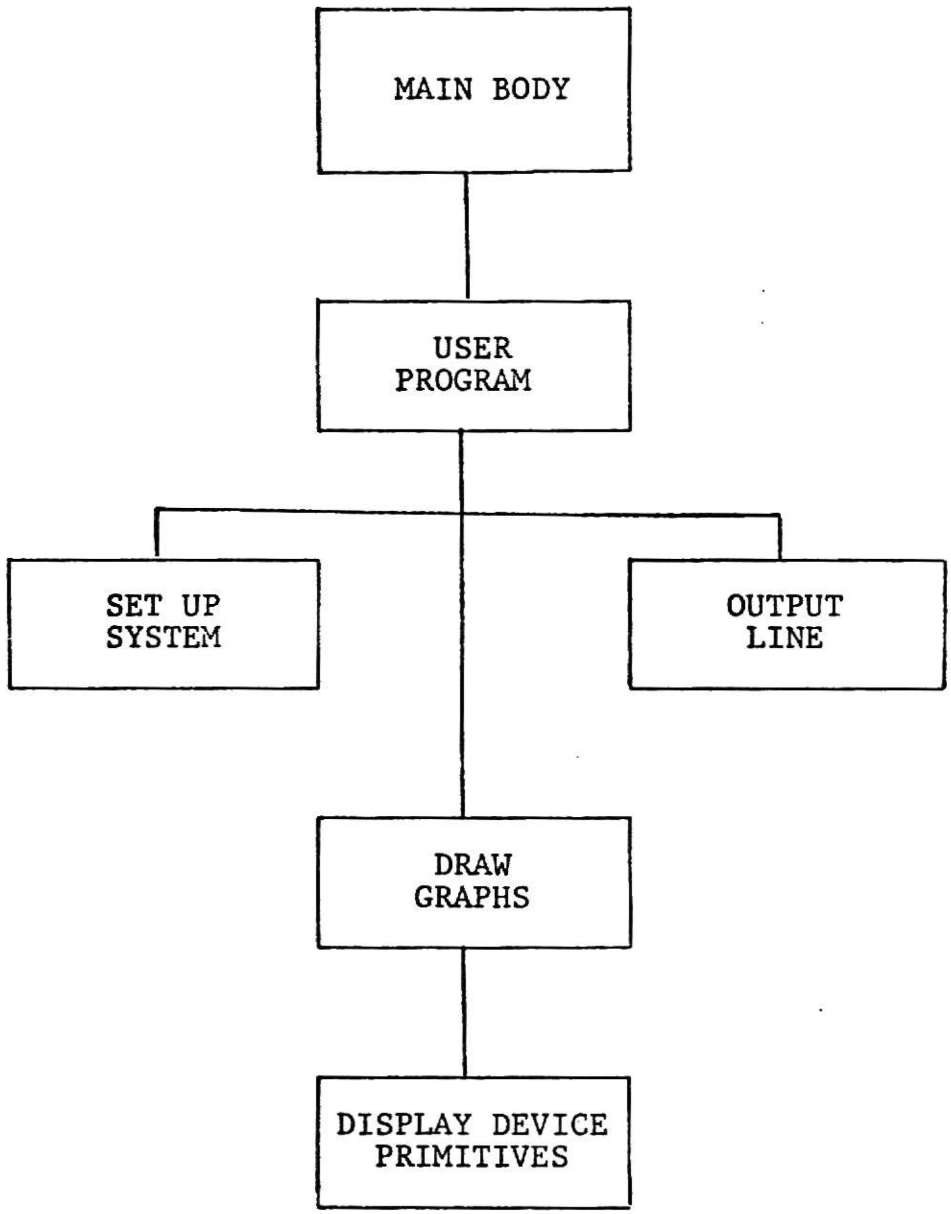

Figure 8. Chromatics Display Device Schematic.

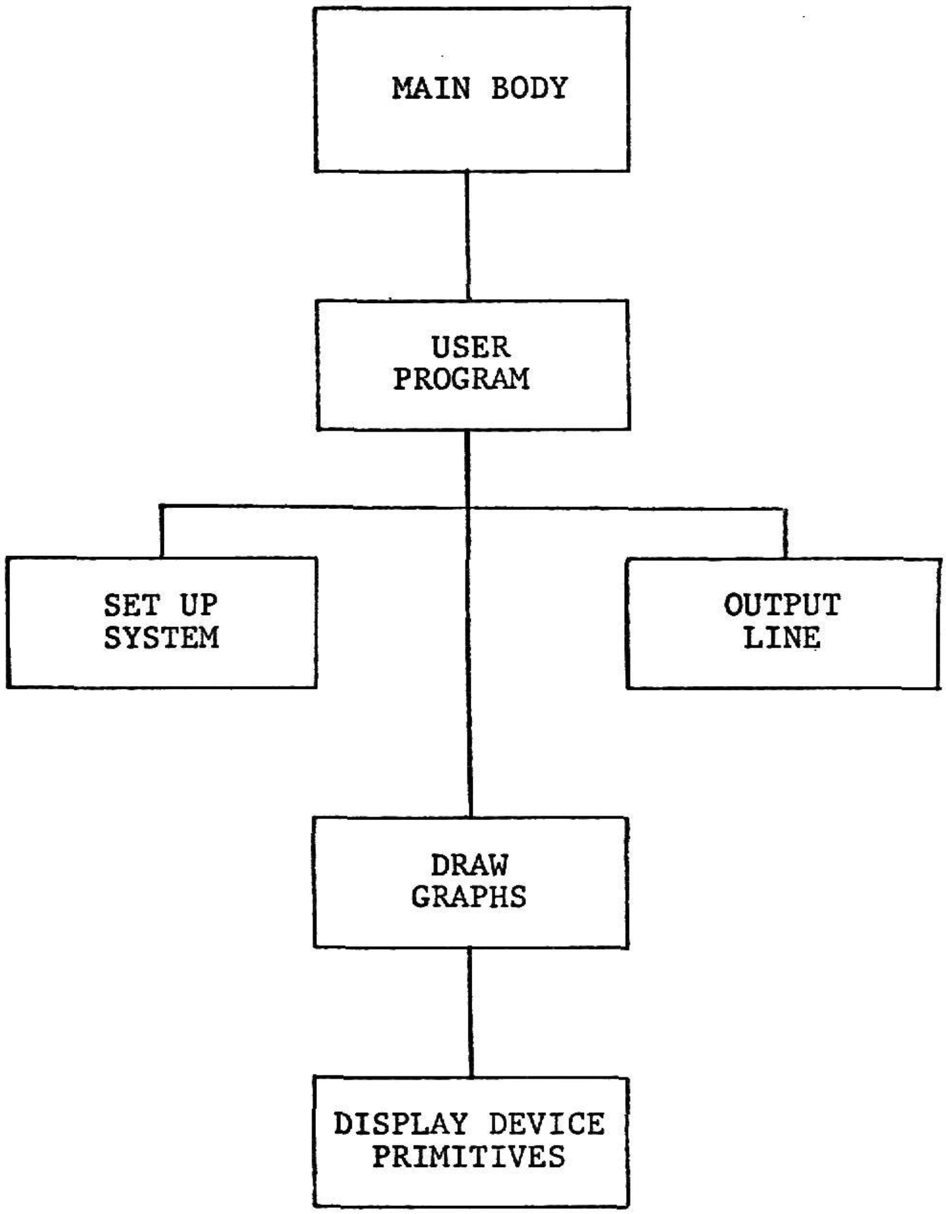

Figure 9. Plotter Display Device Schematic.

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.

2. The number of testing intervals.

3. The length of each testing interval.

4. The number of software errors discovered to date in the testing process.

5. As assistance indicator used to cause the assistance procedure to present amplifying instructions pertaining to a particular question.

Outputs of this procedure are text type queries made to the CRT screen.

b. Procedure Partial Prompting. This procedure performs the identical functions of the previous procedure with the exception of the degree of explanation provided. All access to global data variables, procedure invocations, execution sequence, and output is done in the same manner.

c. Procedure Compute Schick-Wolverton. This procedure performs the computations of the data in accordance with the Schick-Wolverton model. This procedure is invoked from either of the two prompting procedures, with no parameters. The procedure's span of control, or access with respect to other procedures, is presented in Appendix B. There are no outputs from this procedure.

The procedure exercises access to global data variables used to indicate the following:

1. The computed reliability.

2. The computed constant of proportionality.

3. The computed mean time to failure.

4. The computed standard deviation.

5. The computed time to discover all errors.

6. The number of errors initially present in the software package.

7. The number of testing intervals.

8. The length of each testing interval.

9. The number of errors discovered to date in the testing process.

d. Procedure Compute Jelinski-Moranda. This procedure performs the same functions in the same manner as the previous procedure, with the obvious exception that it is for the Jelinski-Moranda model. Span of control, invocation, outputs, access to global data variables are as defined for the Schick-Wolverton computational procedure.

e. Procedure Square Root. This procedure is used to calculate a square root. It is invoked from either of the two model computational procedures, with the input parameter of the value for which the root is desired. This procedure has no access with respect to other procedures or global data variables and performs no output functions.

f. Procedure Load Tabular Display S-W. This procedure performs the process of initializing the variables used in the Schick-Wolverton portion of the tabular form of solution. This procedure is invoked from either of the two prompting procedures, with no input parameters. The procedure has no access with respect to other procedures and performs no output functions.

The procedure exercises access to global data variables used to indicate the following:

1. The computed reliability.

2. The computed constant of proportionality.

3. The computed mean time to failure.

4. The computed standard deviation.

5. The computed time to discover all errors.

6. The data variables used to represent the above variables in the solution form.

g. Procedure Load Tabular Display J-M. This procedure performs the same function as the previous procedure for the Jelinski-Moranda portion of the tabular solution form. The invocation, span of control with respect to other procedures, and output functions are also the same.

The procedure exercises access to the same type of global data variables as the preceding procedure but obviously pertaining to the Jelinski-Moranda model.

h. Procedure Print Tabular Display. This procedure is used to output the tabular solution to the CRT face. The procedure is invoked from either of the two prompting procedures, with the input parameter of which model was selected for use. The span of control with respect to other procedures is presented in Appendix B.

The procedure exercises access to the global data variables of the tabular display. These are the variables defined by the two previous loading procedures.

i. Procedure Write Integer. This procedure is used to print an integer value to the CRT face. The procedure is invoked by Procedure Print Tabular Display, with an input parameter of a line of text. The procedure accesses no other procedures or global data variables.

j. Procedure Write Tabular Solution. This procedure provides the textual information of the tabular solution on the CRT. The procedure is invoked from Procedure Print Tabular Solution, with an input parameter of a text string. The procedure exercises no access to other procedures or global data variables.

k. Procedure Collect Graph Information. This procedure performs the function of querying the user as to where the graphical solution is to be displayed. The procedure is invoked from either of the two prompting procedures, with no input parameters. The procedure's span of control with respect to other procedures is presented in Appendix B. The procedure produces no ouput.

The procedure exercises access to the following global data variables used to indicate the following:

1. The destination of the graphical solution.

2. As assistance indicator used to cause the assistance procedure to present amplifying instructions.

l. Procedure Load Graph Data S-W. This procedure provides the computational function of converting the computed reliability and the computed mean time to failure of the Schick-Wolverton model to coordinate points for display on the selected display device. The procedure is invoked from either of the two prompting procedures, with the input parameter of the scale selected for the mean time to failure axis of the graphical solution. This procedure exercises no control over other procedures and outputs nothing.

The procedure exercises access to the following global data variables used to indicate the following:

1. The computed reliability.

2. The computed mean time to failure.

3. The number of errors discovered to date in the testing process.

4. The scale selected for the mean time to failure axis.

5. The data elements necessary to hold the coordinate points for subsequent plotting by the display driver program.

m. Procedure Load Graph Data J-M. This procedure performs exactly the same function as the previous procedure, with the obvious exception that it deals with the Jelinski-Moranda model. The span of control with respect to other procedures and global data variables is also the same. The procedure is invoked by either of the two prompting procedures, with the input parameter of the scale selected for the mean time to failure axis of the graphical solution.

n. Procedure Help. This procedure provides amplifying instructions to the user, based on the input parameter of the assistance control flag. This procedureis invoked from either of the two prompting procedures or the Collect Graph Information procedure. The procedure accesses control over only the assistance flag. The procedure's span of control over other procedures is limited to the procedure used to write a string of textual information to the CRT screen.

o. Procedure Write String. This procedure provides the implementation with the function of presenting textual information on the CRT screen. This procedure is invoked from either of the two prompting procedures, Procedure Collect Graph Information, Procedure Help,and the main body, with an input parameter of a text string. The procedure exercises no access over other procedures or global data variables.

p. Procedure Read Integer. This procedure is used to read integer input values from the CRT face. The procedure is invoked from either of the two prompting procedures, with an input parameter of the number read. This procedure exercises no access to other procedures or global data variables.

Q. Procedure Draw Graphs. This procedure is used to control the execution and invocation of the display driver programs. The procedure is invoked from either of the two prompting procedures. The procedure's span of control with respect to other procedures is presented in Appendix B. The procedure does not output and exercises access to the global data variable used to indicate the destination of the graphical solution, or where the graphical solution is to be presented.

Procedures of the display driver programs are as follows. Additional details of each procedure are presented in

Appendix I.

a. Procedure User Program. This procedure is used to control the execution sequence of the respective display device program. The procedure is invoked by the main program, with an input parameter of the coordinates to be plotted. The procedure's span of control with respect to other procedures is presented in Appendix B. The procedure exercises no access to global data variables and does no output functions.

b. Procedure Set Up the System. This procedure is used to draw the graphical framework of the graphical solution, label each axis of the graph, and provide the necessary notes for user understanding of the presented graph. The procedure exercises no access with respect to other procedures and accesses only the global data variable of the scale selected for the mean time to failure axis. The procedure is invoked from the previous procedure.

c. Procedure Draw Graph. This procedure performs the plotting of the coordinates on the graphical framework. It is invoked from the Procedure User Program, with no input parameters. The procedure exercises no access with respect to other procedures and exercises access to only the global data variable used to contain the graph coordinate points to be plotted.

Both display driver programs of this implementation contain the above three procedures. The function of the procedures is identical in each driver program. Additionally, each driver program contains several procedures used to invoke the primitives of the respective display device. The procedures are not listed in this documentation but may be referenced in the user manuals associated with the respective display device.

APPENDIX D

ISSUES OF MODIFICATION

As mentioned in the first chapter of this report, one goal of this implementation was to facilitate future modifications. During the design phase of the implementation, I realized that there would exist a great potential to expand this implementation, particularly in the addition of display devices. For that reason, functions of the program were encapsulated into specific procedures, almost on a one-to-one basis (see Appendix C for detailed information pertaining to procedural functions). As an example of this approach, the process of loading the tabular solution form for each of the two models of the implementation was divided into two procedures, one for each model. Additionally, the process of loading the data to be displayed graphically for each of the two models was also divided into separate procedures.

I approached other classical issues of program modification through the facilities of the PASCAL programming language itself. These issues are, in part, addressed below.

a. Array Declarations. Arrays are declared with the upper bound as a constant. Any change to these array sizes can then be accomplished through changing only one constant. Arrays of the implementation that are candidate for future modification are the two arrays that contain the respective model's error interval length and the respective model's graphical x and y coordinates. See Appendix I for more detailed information concerning arrays and their use in the implementation programs.

b. PASCAL For Loop Limits. All "for" loops used within the programs are specified with a maximum range as a constant or a variable value, not a specific value. As in 'a' above, changes to the upper limits of the loops can then be made through the modification of the appropriate constant. See Appendix I for more detailed information concerning the use of For Loops within the implementation programs.

c. Special Values Used in Computations. Throughout the program, these types of values are referenced, almost exclusively in comparisons or computations. Examples are in the process of calculating the y coordinates for the mean time to failure values of a respective model, a constant is used to denote the origin of the graph and another constant is used to denote the number of pixels per unit of the particular device. A final example occurs in the computation of square roots. The accuracy of the root produced is described in terms of a constant. Any change to accuracy desired in the square root procedure or graphical presentation on a display device can then be more easily changed by simply changing the values of their associated constants.

The modification issue of supporting additional display drive programs, and thus additional display devices, is somewhat straight forward. As explained in detail in Appendix I, the interface between the main program of this implementation and the display driver programs is an external one. The addition of a display device would then require a separate display driver program and the invocation of this program from the main program.

There are several other important issues of modification that should also be addressed. These are the modification of the display driver program interface, the modification of the display drivers themselves, and a modification to the graphical solution form.

Concerning the issue of the display driver program interface, this implementation treats all display driver programs as external to the main program, in the sense of an external procedure call. This interface was chosen to restrict the size of the main driver program and because of multiple external procedure invocations from the display driver programs themselves. The alternate interface capability investigated during the design of the implementation did not suppo-t the external calls of the display driver programs and thus was eliminated, not without much effort, however. Therefore, to change the interface with the existing display driver programs, the driver programs themselves would need to -e modified to remove these external procedure invocations. Examples of invocations made externally from the driver programs are trigonometric function calls used in drawing routines of each display driver program. See Appendix I for additional information and examples concerning the external invocations of these display driver programs.

The next area of modification that should be addressed is in the display driver program modification. These programs were already developed to perform those primitive functions of plotting and were modified by me to produce the particular results that were needed in this implementation. These results were to accept the graphical data from the main program, to draw the initial graphical framework, to label the graphical framework, and to draw the graphs themselves. Unfortunately, these existing driver programs are not flexible in the classical issues of modification discussed earlier. Those procedures added to perform the functions just mentioned do follow the guidelines I set down for modification support in the main program however.

The final area of modification that should be addressed is the graphical solution form itself. The present solution form, as shown in Appendix H, presents the relationship between reliability and mean time to failure for each interval of the software project. AS this relationship is concerned with each preceeding interval, this often causes wide fluctuations in the values displayed on the graph. The proposed modification would be to depict the relationship between reliability and mean time to failure for an interval of constant length. The interval length to be displayed would be selected by the user before the graphical solution is presented.

APPENDIX E

SCHICK-WOLVERTON SOFTWARE RELIABILITY MODEL

The Schick-Wolverton Reliability Model was developed by George J. Schick and Raymon W. Wolverton. The model computes a reliability estimate and estimates of the mean time to failure and total time necessary to discover all errors within a software package.

Estimates produced by the model are statistically based upon the following hazard function [13]:

$$Z(t_i) = \phi(N-(i-1))t_i$$

where ϕ is a constant of proportionality calculated for each iteration of the model and used to keep the area under the reliability curve equal to one.

N is the number of errors initially present in the software package. This value can be estimated or calculated.

t_i is the time between the i^{th} and $(i-1)^{st}$ error discovery. This is often referred to as the i^{th} debugging interval.

i is a specific discovered error.

Additionally, the model is based on the following four assumptions[2,8,13,16]:

a. The amount of time between error occurrences is statistically modeled by a Rayleigh distribution.

b. The error occurrence rate is proportional to the number of errors remaining in the system and the time spent in error discovery (debugging) testing.

c. Only one error is discovered in each debugging interval.

d. Each error discovered is immediately removed, reducing the number of errors remaining by one.

The reliability model has the following form [17]:

$$R(t_i) = \exp(-\phi(N-(i-1))t_i^2/2)$$

The estimated time to discover all remaining errors of the software package has the following form [17]:

$$TDRE = \left[\sum_{i=1}^{N-n} {}^{1}/_{i}\right] / \phi$$

where n is the number of errors discovered to date and the remaining variables are as defined previously.

The estimated mean time to failure of the system has the following form [17]:

$$MTTF = \left[\pi / (2 \cdot \phi \cdot [N-n])\right]^{\frac{1}{2}}$$

Finally, the constant of proportionality is calculated by the following form [13]:

$$\hat{\phi} = \left[\sum_{i=1}^{n} \frac{2}{N-(i-1)}\right] \cdot \left[\frac{1}{\sum_{i=1}^{n} t_i^2}\right]$$

APPENDIX F

JELINSKI-MORANDA SOFTWARE RELIABILITY MODEL

The Jelinski-Moranda Reliability Model was developed by Z. Jelinski and P. B. Moranda. The model computes the reliability estimate and estimates for the total time required to discover all remaining errors and the mean time to failure of the software package.

Estimates produced by the model are based statistically on the following hazard function [7,14,17]:

$$Z(t_i) = \phi(N-(i-1))$$

where ϕ is a constant of proportionality calculated for each iteration of the model and used to keep the area under the reliability curve to one.

N is the number of errors initially present in the software package. This value can be estimated or calculated.

i is a specific discovered error.

Additionally, the model is based on the following five assumptions [1,2,4,9]:

a. The amount of debugging time between error occurrences has an error occurrence rate proportional to the number of errors remaining.

b. Each error discovered is immediately removed, reducing the number of errors remaining in the software package by one.

c. The occurrence rate between errors is constant.

d. All errors remaining in the program at any given time are equally likely to occur.

e. Only one error is discovered in each debugging interval.

The reliability model has the following form [17]:

$$R(t_i) = \exp(-\phi(N-(i-1))t_i)$$

where t_i is the time between the i^{th} and $(i-1)^{st}$ error discovery. This is often referred to as the i^{th} debugging interval.
and other variables are as defined previously.

The estimated time to discover all remaining errors of the software package has the following form [17]:

$$TDRE = \left[\sum_{i=1}^{N-n} 1/i \right] / \phi$$

where n is the number of errors discovered to date and the remaining variables are as defined previously.

The estimated mean time to failure of the system has the following form [17]:

$$MTTF = {}^{1}/(\phi \cdot (N-n))$$

Finally, the constant of proportionality is calculated by the following form [13]:

$$\hat{\phi} = \frac{n}{NT - \sum_{i-1}^{n} (i-1)\, t_i}$$

where T is defined to be the summation of the length of each debugging interval, or $T = \sum_{i=1}^{n} t_i$ and all other variables are as defined previously.

APPENDIX G

VALIDATION OF PROGRAM MODEL COMPUTATIONS

INPUT DATA		
Initial Errors	= 50	
Number of Errors Discovered	= 5	
Number of Testing Intervals	= 5	
Length of Respective Intervals	= 1,2,3,4,5	

SOLUTIONS FOR SCHICK-WOLVERTON		
HAND CALCULATED		PROGRAM CALCULATED
.205	Reliability	.21
.0070	Constant of Proportionality	.007
3.155	MTTF	3
180.866	Time to Discover Remaining Errors	180

SOLUTIONS FOR JELINSKI-MORANDA		
HAND CALCULATED		PROGRAM CALCULATED
.113	Reliability	.11
.00379	Constant of Proportionality	.0038
3.034	MTTF	3
1159.25	Time to Discover Remaining Errors	1159

Table 1. Comparison of Model Accuracy - Case 1.

INPUT DATA		
Initial Errors		= 100
Number of Errors Discovered		= 20
Number of Testing Intervals		= 20
Length of Respective Intervals		= 1,1,1,1,1,1,1,1 1,1,1,1,1,1,1,1 1,1,1,1
SOLUTIONS FOR SCHICK-WOLVERTON		
HAND CALCULATED		PROGRAM CALCULATED
.409	Reliability	.41
.02221	Constant of Proportionality	.0222
1.09	MTTF	1
224.38	Time to Discover Remaining Errors	224
SOLUTIONS FOR JELINSKI-MORANDA		
HAND CALCULATED		PROGRAM CALCULATED
.411	Reliability	.41
.01103	Constant of Proportionality	.0110
1.03	MTTF	1
448.87	Time to Discover Remaining Errors	449

Table 2. Comparison of Model Accuracy - Case 2.

INPUT DATA		
Initial Errors		= 5
Number of Errors Discovered		= 4
Number of Testing Intervals		= 4
Length of Respective Intervals		= 1,2,3,4
SOLUTIONS FOR SCHICK-WOLVERTON		
HAND CALCULATED		PROGRAM CALCULATED
.249	Reliability	.25
.08561	Constant of Proportionality	.0856
4.11	MTTF	4
12.23	Time to Discover Remaining Errors	12
SOLUTIONS FOR JELINSKI-MORANDA		
HAND CALCULATED		PROGRAM CALCULATED
.5905	Reliability	.59
.00738	Constant of Proportionality	.0074
3.04	MTTF	3
8.21	Time to Discover Remaining Errors	8

Table 3. Comparison of Model Accuracy - Case 3.

INPUT DATA		
Initial Errors		= 90
Number of Errors Discovered		= 27
Number of Testing Intervals		= 27
Length of Respective Intervals		= 1,1,1,2,2,2,3,3,3 4,4,4,5,5,5,6,6,6 7,7,7,8,8,8,9,9,9
SOLUTIONS FOR SCHICK-WOLVERTON		
HAND CALCULATED		PROGRAM CALCULATED
.123	Reliability	.12
.00078	Constant of Proportionality	.0008
5.02	MTTF	5
5705.19	Time to Discover Remaining Errors	5705
SOLUTIONS FOR JELINSKI-MORANDA		
HAND CALCULATED		PROGRAM CALCULATED
.209	Reliability	.21
.00272	Constant of Proportionality	.0027
5.989	MTTF	6
1726.15	Time to Discover Remaining Errors	1726

Table 4. Comparison of Model Accuracy - Case 4.

INPUT DATA		
Initial Errors	= 8	
Number of Errors Discovered	= 1	
Number of Testing Intervals	= 1	
Length of Respective Intervals	= 1	

SOLUTIONS FOR SCHICK-WOLVERTON		
HAND CALCULATED		PROGRAM CALCULATED
.369	Reliability	.37
.2503	Constant of Proportionality	.25
1.02	MTTF	1
9.87	Time to Discover Remaining Errors	10

SOLUTIONS FOR JELINSKI-MORANDA		
HAND CALCULATED		PROGRAM CALCULATED
.4204	Reliability	.42
.1253	Constant of Proportionality	.1250
1.01	MTTF	1
20.76	Time to Discover Remaining Errors	21

Table 5. Comparison of Model Accuracy - Case 5.

APPENDIX H

SOLUTION FORMATS

```
- TABULAR DISPLAY
-
-
- SCHICK-WOLVERTON DATA IS AS FOLLOWS.
-
- RELIABILITY IS 0.37
- STANDARD DEVIATION IS 1
- CONSTANT OF PROPORTIONALITY IS 0.10000
- MEAN TIME TO FAILURE IS 1
- TIME TO DISCOVER ALL ERRORS IS 1
-
-
- JELINSKI-MORANDA DATA IS AS FOLLOWS.
-
- RELIABILITY IS 0.40
- STANDARD DEVIATION IS 1
- CONSTANT OF PROPORTIONALITY IS 0.10000
- MEAN TIME TO FAILURE IS 1
- TIME TO DISCOVER ALL ERRORS IS 1
-
-
-  PRESS THE RETURN KEY WHEN YOU ARE FINISHED
   READING THIS PAGE
```

Figure 10. Tabular Solution Format Example (Both Models Selected).

```
- TABULAR DISPLAY
-
-
- SCHICK-WOLVERTON DATA IS AS FOLLOWS.
-
- RELIABILITY IS 0.37
- STANDARD DEVIATION IS 1
- CONSTANT OF PROPORTIONALITY IS 0.10000
- MEAN TIME TO FAILURE IS 1
- TIME TO DISCOVER ALL ERRORS IS 1
-
-
-  PRESS THE RETURN KEY WHEN YOU HAVE FINISHED
   READING THIS PAGE
```

Figure 11. Tabular Solution Format Example (Schick-Wolverton Model Selected).

```
- TABULAR DISPLAY
-
-
- JELINSKI-MORANDA DATA IS AS FOLLOWS.
-
- RELIABILITY IS 0.37
- STANDARD DEVIATION IS 1
- CONSTANT OF PROPORTIONALITY IS 0.10000
- MEAN TIME TO FAILURE IS 1
- TIME TO DISCOVER ALL ERRORS IS 1
-
-
-  PRESS THE RETURN KEY WHEN YOU HAVE FINISHED
   READING THIS PAGE
```

Figure 12. Tabular Solution Format Example (Jelinski-Moranda Model Selected).

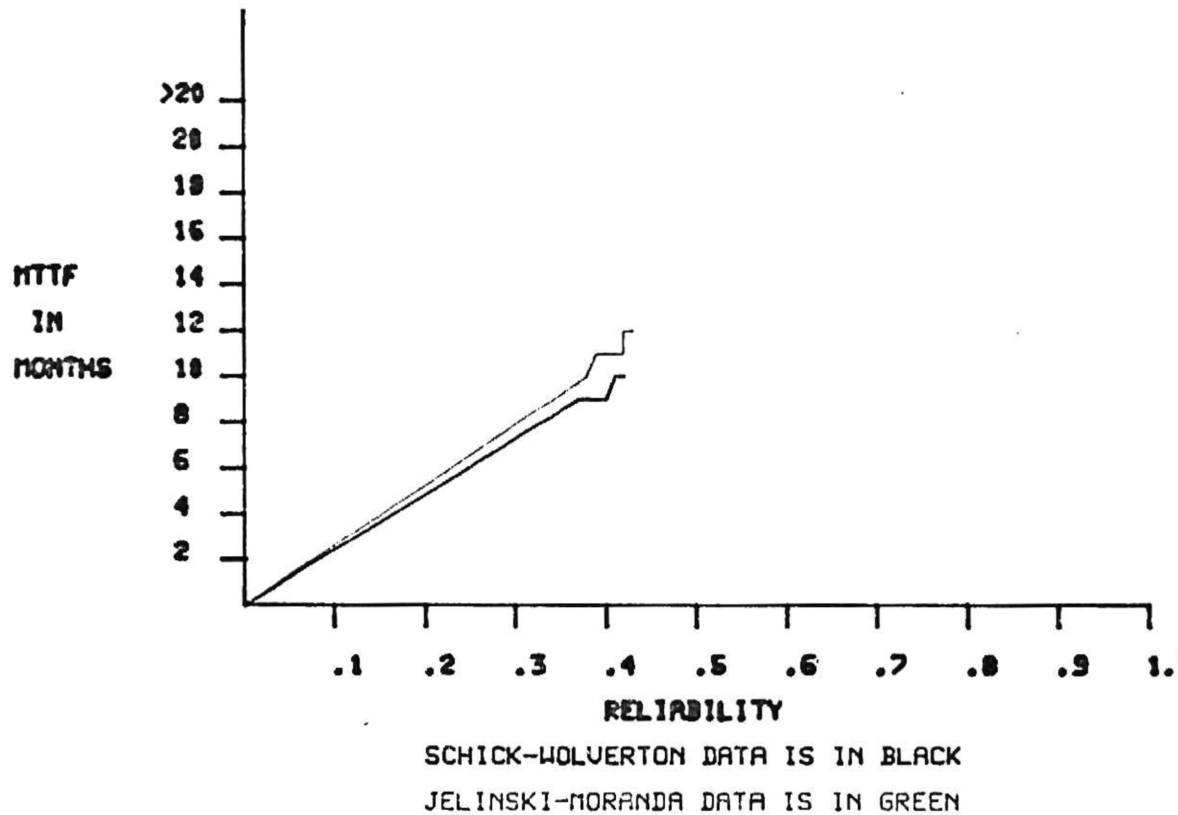

Figure 13. Graphical Output Example.

APPENDIX I

PROGRAM SOURCE CODE LISTINGS

```
  "*******************************************************"
  "
"
  "  THIS  PROGRAM  COMPUTES  THE RELIABILITY
AND THE     "
  "  MEAN  TIME  TO  FAILURE  OF  A PARTIALLY
DEBUGGED      "
  "   SOFTWARE  PACKAGE.    THE   PROGRAM  IS
INTERACTIVE IN  "
  "  NATURE  AND  WAS  PREPARED  TO  MEET THE
REQUIREMENT   "
  "  OF A REPORT FOR THE DEGREE OF MASTER  OF
SCIENCE   "
  "        IN        COMPUTER        SCIENCE.
"
  "
"
  "
"
  "         THIS  PROGRAM  WAS  PREPARED  BY
"
  "
"
  "                 JOHNNIE    O.     RANKIN
"
  "
"
  "                   1    DECEMBER     1981
"
  "
"
  "            AT  KANSAS  STATE  UNIVERSITY
"
  "
"
  "         MAJOR ADVISOR  FOR  THIS PROJECT
"
  "
"
  "                                      WAS
"
  "
"
  "             DR.   DAVID   A.  GUSTAFSON
"
  "
"
  "*******************************************************"
```

```
"KANSAS STATE UNIVERSITY" "DEPARTMENT OF
COMPUTER SCIENCE"
CONST COPYRIGHT = 'COPYRIGHT ROBERT YOUNG
1978';
      " THE FOLLOWING PROCEDURES ARE A HEADER
REQUIRED BY THE "
      " INTERDATA PASCAL COMPILER IN USE IN
FAIRCHILD HALL    "
"########### # PREFIX # ###########" CONST
NL = '(:10:)';    FF = '(:12:)';    CR =
'(:13:)';   EM = '(:25:)'; CONST PAGELENGTH =
512; TYPE PAGE = ARRAY (.1..PAGELENGTH.) OF
CHAR; CONST LINELENGTH = 132; TYPE LINE =
ARRAY (.1..LINELENGTH.) OF CHAR; CONST
IDLENGTH = 12; TYPE IDENTIFIER = ARRAY
(.1..IDLENGTH.) OF CHAR; TYPE FILE = 1..2;
TYPE FILEKIND = (EMPTY, SCRATCH, ASCII,
SEQCODE, CONCODE); TYPE FILEATTR = RECORD
                  KIND: FILEKIND;
                  ADDR: INTEGER;
                  PROTECTED: BOOLEAN;
                  NOTUSED: ARRAY (.1..5.) OF
INTEGER
                END; TYPE IODEVICE =
  (TYPEDEVICE,    DISKDEVICE,    TAPEDEVICE,
PRINTDEVICE, CARDDEVICE); TYPE IOOPERATION =
(INPUT, OUTPUT, MOVE, CONTROL); TYPE IOARG =
(WRITEEOF, REWIND, UPSPACE, BACKSPACE); TYPE
IORESULT =
  (COMPLETE,    INTERVENTION,    TRANSMISSION,
FAILURE,
   ENDFILE, ENDMEDIUM, STARTMEDIUM); TYPE
IOPARAM = RECORD
                 OPERATION: IOOPERATION;
                 STATUS: IORESULT;
                 ARG: IOARG
               END;    TYPE    TASKKIND    =
(INPUTTASK, JOBTASK, OUTPUTTASK); TYPE ARGTAG
=
  (NILTYPE,   BOOLTYPE,    INTTYPE,    IDTYPE,
PTRTYPE); TYPE POINTER = @BOOLEAN;
TYPE PASSPTR = @PASSLINK;
TYPE PASSLINK = RECORD
  OPTIONS: SET OF CHAR;
  FILLER1: ARRAY(.1..7.) OF INTEGER;
  FILLER2: BOOLEAN;
  RESET_POINT: INTEGER;
  FILLER3: ARRAY (.1..11.) OF POINTER
  END;
TYPE ARGTYPE = RECORD
                 CASE TAG: ARGTAG OF
                   NILTYPE, BOOLTYPE: (BOOL:
BOOLEAN);
                   INTTYPE: (INT: INTEGER);
```

```
                  IDTYPE: (ID: IDENTIFIER);
                  PTRTYPE: (PTR: PASSPTR)
              END; CONST  MAXARG  =  10; TYPE
ARGLIST = ARRAY (.1..MAXARG.) OF ARGTYPE;
```

```
   "*************************************************************
   "*
*"
   "* INSTANCES  OF  THE  FOLLOWING RECORD ARE
USED TO HOLD DATA *"
   "*  APPLICABLE  TO  THE  GRAPHICAL  DISPLAY
DEVICE WHICH MAY BE *"
   "* SELECTED  FOR  USE  WITHIN THIS PROGRAM.
EACH PROGRAM      *"
   "* CONTROLLING  THESE  DISPLAY  DEVICES  IS
TREATED AS AN EX-  *"
   "*  TERNAL  PROCEDURE  AND  IS  PASSED  THE
RESPECTIVE INSTANCE  *"
   "* OF THIS RECORD.  INSTANCES USED IN  THIS
PROGRAM ARE      *"
   "* CHROM  DATA,  USED  FOR  THE  CHROMATICS
DEVICE, PLOT DATA,  *"
   "* USED FOR  THE  PLOTTER  DEVICE, AND SPIN
DATA, USED FOR THE*"
   "*        SPIN        WRITER        DEVICE.
*"
   "*
*"
   "*************************************************************


TYPE COORDINATE = RECORD
                    S_W_PLOT
: ARRAY(.1..250,1..2.) OF INTEGER;
                    J_M_PLOT
: ARRAY(.1..250,1..2.) OF INTEGER;
                    COMBINATION_SELECTED
: INTEGER;
                    SCALE_DESIRED
: INTEGER;
                    NUMBER_OF_ERRORS_OBSERVED
: INTEGER
                  END;
```

```
TYPE ARGSEQ = (INP, OUT); TYPE PROGRESULT =
  (TERMINATED,     OVERFLOW,     POINTERERROR,
RANGEERROR, VARIANTERROR,
   HEAPLIMIT,       STACKLIMIT,       CODELIMIT,
TIMELIMIT, CALLERROR); PROCEDURE  READ(VAR C:
CHAR); PROCEDURE  WRITE(C:  CHAR);  PROCEDURE
OPEN(F:  FILE;  ID:  IDENTIFIER;  VAR   FOUND:
BOOLEAN); PROCEDURE CLOSE(F: FILE);  PROCEDURE
GET(F:  FILE;  P:  INTEGER;  VAR  BLOCK:  UNIV
PAGE); PROCEDURE PUT(F: FILE; P: INTEGER;  VAR
BLOCK: UNIV PAGE);  FUNCTION  LENGTH(F: FILE):
INTEGER;  PROCEDURE  MARK(VAR  TOP:  INTEGER);
PROCEDURE  RELEASE(TOP:  INTEGER);   PROCEDURE
IDENTIFY(HEADER: LINE);  PROCEDURE  ACCEPT(VAR
C:   CHAR);   PROCEDURE   DISPLAY(C:    CHAR);
PROCEDURE   NOTUSED;    PROCEDURE    NOTUSED2;
PROCEDURE   NOTUSED3;   PROCEDURE    NOTUSED4;
PROCEDURE   NOTUSED5;   PROCEDURE    NOTUSED6;
PROCEDURE   NOTUSED7;   PROCEDURE    NOTUSED8;
PROCEDURE   NOTUSED9;   PROCEDURE   NOTUSED10;
PROCEDURE  RUN(ID:   IDENTIFIER;   VAR  PARAM:
COORDINATE;
              VAR LINE:  INTEGER;  VAR RESULT:
PROGRESULT);
```

```
"#################################################"
     PROGRAM RELIABILITY_MODEL (PARAM:  LINE);
"#################################################"


"####################################################"
"#
#" "# THIS PROGRAM COULD NOT HAVE BEEN
COMPLETED WITH-#" "# OUT THE ABLE ASSISTANCE
OF MR. ROBERTY YOUNG OF #" "# THE COMPUTER
SCIENCE DEPARTMENT, CARDWELL HALL, #" "# AND
MR. CARLOS QUALLS, DEPARTMENT OF COMPUTER   #"
"# SCIENCE, FAIRCHILD HALL. MR. YOUNG'S
EXPERTISE #" "# PROVED TO BE INVALUABLE TO ME
IN THE AREA OF   #" "# ESTABLISHING THE
INTERFACE BETWEEN THE INTERDATA#" "# AND THE
REMOTE DISPLAY DEVICES. MR. QUALLS WAS #" "#
AN IMMENSE HELP THROUGH HIS KNOWLEDGE OF THE
#" "# INTERDATA SYSTEM, ESPECIALLY TROUBLE
SHOOTING #" "# AND DEBUGGING TECHNIQUES.
#"                                          "#
#"
"####################################################"


    CONST
      MAX_ALLOWED                               =
250;
      FULL_LEVEL                                =
'F';
      PARTIAL_LEVEL                             =
'P';
      NEED_HELP                                 =
'H';
      DO_NOT_NEED_HELP                          =
'N';
      BLANK                                   = '
';


    VAR
      INITIAL_ERRORS                            :
INTEGER;
      TOTAL_INTERVALS                           :
INTEGER;
      ERRORS_DISCOVERED                         :
INTEGER;
      DESTINATION                               :
INTEGER;
      I                                         :
INTEGER;
      J                                         :
INTEGER;
      HELP_FLAG                                 :
```

```
INTEGER;
      TAB_S_W_RELIABILITY                      :
INTEGER;
      TAB_S_W_MTTF                             :
INTEGER;
      TAB_S_W_TIME_TO_FIND_ALL_ERRORS          :
INTEGER;
      TAB_S_W_STD                              :
INTEGER;
      TAB_S_W_PHI                              :
INTEGER;
      TAB_J_M_RELIABILITY                      :
INTEGER;
      TAB_J_M_MTTF                             :
INTEGER;
      TAB_J_M_TIME_TO_FIND_ALL_ERRORS          :
INTEGER;
      TAB_J_M_STD                              :
INTEGER;
      TAB_J_M_pHI                              :
INTEGER;

      J_M_TIME_TO_FIND_ALL_ERRORS              :
REAL;
      S_W_TIME_TO_FIND_ALL_ERRORS              :
REAL;
      J_M_STANDARD_DEVIATION                   :
REAL;
      S_W_STANDARD_DEVIATION                   :
REAL;
      S_W_PHI                                  :
REAL;
      J_M_PHI                                  :
REAL;
      ANSWER                                   :
REAL;
      S_W_RELIABILITY                          :
REAL;
      S_W_MTTF                                 :
REAL;
      J_M_RELIABILITY                          :
REAL;
      J_M_MTTF                                 :
REAL;

      MESSAGE_IN                               :
CHAR;
      DUMMY                                    :
CHAR;

      SUCCESSFUL_INPUT                         :
BOOLEAN;

      J_M_DATA_1                               :
```

```
ARRAY(.1..MAX_ALLOWED.) OF INTEGER;
      S_W_DATA_1                                      :
ARRAY(.1..MAX_ALLOWED.) OF INTEGER;

      CHROM_DATA                                      :
COORDINATE;
      PLOT_DATA                                       :
COORDINATE;
      SPIN_DATA                                       :
COORDINATE;


      PROCEDURE CHROFIX(PARAM : COORDINATE);
        EXTERN;


      PROCEDURE PLOTFIX(PARAM : COORDINATE);
        EXTERN;


      PROCEDURE SPINFIX(PARAM : COORDINATE);
        EXTERN;


      FUNCTION DEXP(VALUE : REAL) : REAL;
        FORTRAN;
```

```
PROCEDURE WRITE_STRING(TEXT : LINE);


  "#############################################################
  "#
#"
  "# THIS PROCEDURE  IS  USED  TO OUTPUT CONVERSATIONAL TYPE
#"
  "# MESSAGES TO THE  USER  AT  THE CRT FACE.  IT IS INVOKED
#"
  "# FROM SEVERAL PLACES  WITHIN  THE PROGRAM, BUT PRIMARILY
#"
  "# FROM  THE  PROMPTING  PROCEDURES  AND THE COLLECT GRAPH
#"
  "#                 INFORMATION                  PROCEDURE.
#"
  "#
#"
  "#############################################################


  VAR
    I  : INTEGER;

    BEGIN "PROCEDURE WRITE STRING"
      I := 1;
      WHILE(TEXT(.I.) <> '(:0:)')DO
        BEGIN
          DISPLAY(TEXT(.I.));
          I := SUCC(I)
        END;
      DISPLAY(NL)
    END; "PROCEDURE WRITE STRING"
```

```
PROCEDURE WRITE_INTEGER(I : INTEGER);

    "*******************************************************
    "*
*"
    "* THIS PROCEDURE  IS  USED  TO PRINT INTEGER NUMBERS
*"
    "* ON  THE  TABULAR  DISPLAY  SOLUTION  FORM.   IT IS
*"
    "*  INVOKED   FROM   THE   PROCEDURE   PRINT  TABULAR
DISPLAY.*"
    "*
*"
    "*******************************************************


  CONST
    BLANK               = ' ';
    ASTERISK            = '*';

  VAR
    DIGIT               : INTEGER;
    INT                 : INTEGER;
    J                   : INTEGER;
    INTSTRING           : ARRAY(.1..11.) OF CHAR;

    BEGIN "PROCEDURE WRITE INTEGER"
      DIGIT             := 0;
      INT               := ABS(I);
      J                 := 0;
      FOR J := 1 TO 11 DO
        INTSTRING(.J.) := BLANK;
      J := 11;
      REPEAT
        DIGIT := INT MOD 10;
        INTSTRING(.J.) := CHR(DIGIT + 48);
        INT := INT DIV 10;
        J := PRED(J)
      UNTIL ((INT = 0) OR (J = 0));
      IF I < 0
        THEN INTSTRING(.J.) := ASTERISK;
      FOR J := 1 TO 11 DO
        IF(INTSTRING(.J.) <> BLANK)
          THEN
            DISPLAY(INTSTRING(.J.));
      DISPLAY(NL)
    END; "PROCEDURE WRITE INTEGER"
```

```
PROCEDURE WRITE_TABULAR_SOLUTION(TEXT : LINE);

   "**********************************************************
   "*                                                        *"
   "* THIS PROCEDURE WRITES THE TEXT PORTION OF THE TAB-     *"
   "* ULAR DISPLAY.  IT IS INVOKED FROM THE PROCEDURE        *"
   "* PRINT TABULAR DISPLAY.  THE FUNCTIONING OF THIS        *"
   "* PROCEDURE IS ALMOST IDENTICAL WITH PROCEDURE WRITE_    *"
   "* STRING.                                                *"
   "*                                                        *"
   "**********************************************************"


  VAR
    I  : INTEGER;

    BEGIN "PROCEDURE WRITE TABULAR SOLUTION"
      I := 1;
      WHILE(TEXT(.I.) <> '(:0:)')DO
        BEGIN
          DISPLAY(TEXT(.I.));
          I := SUCC(I)
        END
    END; "PROCEDURE WRITE TABULAR SOLUTION"
```

```
PROCEDURE READ_INTEGER(VAR NUMBER_IN : INTEGER);

  "*******************************************************"
  "*                                                     *"
  "* THIS PROCEDURE PERFORMS THE FUNCTION OF READING     *"
  "* AN INTEGER VALUE WHICH HAS BEEN INPUT BY THE USER   *"
  "* OF THIS PROGRAM.  IT IS INVOKED FROM BOTH OF THE    *"
  "* PROMPTING MODELS.                                   *"
  "*                                                     *"
  "*******************************************************"


  CONST
    MAXINT       = 32767;

  TYPE
    DIGIT        = '0'..'9';

  VAR
    OVERFLOW     : BOOLEAN;
    DUMMY        : INTEGER;
    DIGITS       : SET OF DIGIT;
    C            : CHAR;

    BEGIN "PROCEDURE READ INTEGER"
      DUMMY     := 0;
      NUMBER_IN := 0;
      OVERFLOW  := FALSE;
      DIGITS    := (. .);
      FOR C := '0' TO '9' DO
         DIGITS := DIGITS + (.C.);
      ACCEPT(C);
      WHILE((C IN DIGITS) AND (NOT OVERFLOW))DO
        BEGIN
          DUMMY := ORD(C) - ORD('0');
          IF(NUMBER_IN > (MAXINT - DUMMY)DIV 10)
            THEN
              OVERFLOW := TRUE
            ELSE
              NUMBER_IN := 10 * NUMBER_IN + DUMMY;
          ACCEPT(C)
        END;
      IF(OVERFLOW)
        THEN
          BEGIN
            NUMBER_IN := MAXINT;
            WHILE(C IN DIGITS)DO
              ACCEPT(C)
          END
    END; "PROCEDURE READ INTEGER"
```

```
  PROCEDURE HELP(VAR CALL_FOR_HELP : INTEGER);


     "*******************************************************
     "*
*"
     "* THIS PROCEDURE'S PURPOSE IS TO PROVIDE
ADDITIONAL  *"
     "*  ASSISTANCE  TO  THE  USER  IN  THE
EXECUTION OF THE PRO-*"
     "* GRAM.  IT  IS  INVOKED  FROM THE MAIN
BODY, FROM EACH *"
     "*  PROMPTING  PROCEDURE,  AND  FROM  THE
COLLECT GRAPH    *"
     "*          INFORMATION         PROCEDURE.
*"
     "*
*"
     "*******************************************************


   CONST
     BLANK       = ' ';

   VAR
     DUMMY       : CHAR;
     I           : INTEGER;


    BEGIN "PROCEDURE HELP"


      IF(CALL_FOR_HELP = 1)
        THEN
          BEGIN "CALL FOR HELP = 1"
            WRITE_STRING('THIS PROGRAM  ALLOWS
YOU TO COMPUTE THE  (:0:)');
            WRITE_STRING('RELIABILITY
ASSOCIATED WITH AN ESTIMATE  (:0:)');
            WRITE_STRING('OF   ERRORS   IN   A
SOFTWARE PACKAGE.  THIS  (:0:)');
            WRITE_STRING('COMPUTATION WILL  BE
PERFORMED BY EITHER  (:0:)');
            WRITE_STRING('COMBINATION  OF  TWO
SOFTWARE RELIABILITY  (:0:)');
            WRITE_STRING('MODELS,         THE
SCHICK-WOLVERTON AND/OR THE  (:0:)');
            WRITE_STRING('JELINSKI-MORANDA
MODEL.  DURING EXECUTION  (:0:)');
            WRITE_STRING('YOU WILL RESPOND  TO
A SERIES OF QUESTIONS  (:0:)');
            WRITE_STRING('CONCERNING      YOUR
DESIRES OF THE PROGRAM.   (:0:)');
```

```
            WRITE_STRING('TWO  SOLUTION  FORMS
ARE OFFERED, A GRAPHICAL(:0:)');
            WRITE_STRING('SOLUTION  ON  WHICH
RELIABILITY AND MEAN TIME  (:0:)');
            WRITE_STRING('TO   FAILURE   ARE
PLOTTED AND A TABULAR FORM  (:0:)');
            WRITE_STRING('OF  RELIABILITY  AND
THE MEAN TIME TO FAILURE.  (:0:)');
            WRITE_STRING('YOU  MAY  USE  BOTH
FORMS IF YOU DESIRE.  PRESS  (:0:)');
            WRITE_STRING('THE RETURN KEY  WHEN
YOU ARE FINISHED READ-  (:0:)');
            WRITE_STRING('ING   THIS   PAGE.
(:0:)');
            FOR I := 1 TO 6 DO  DISPLAY(NL);
            ACCEPT(DUMMY);
            FOR I := 1 TO 25 DO  DISPLAY(NL);
            WRITE_STRING('THE   SERIES   OF
QUESTIONS YOU WILL BE ASKED  (:0:)');
            WRITE_STRING('IS INTENDED TO  LEAD
YOU THROUGH A SUCCESSFUL  (:0:)');
            WRITE_STRING('EXECUTION  OF  THE
PROGRAM.  YOU WILL BE ASKED  (:0:)');
            WRITE_STRING('A  GENERAL  QUESTION
AS TO WHICH PROMPTING  (:0:)');
            WRITE_STRING('LEVEL YOU DESIRE  TO
USE DURING EXECUTION  (:0:)');
            WRITE_STRING('OF   THE   PROGRAM.
THERE ARE TWO CHOICES WHICH  (:0:)');
            WRITE_STRING('ARE   AVAILABLE   TO
YOU.  THESE ARE FULL AND  (:0:)');
            WRITE_STRING('PARTIAL   PROMPTING.
FULL PROMPTING REQUIRES   (:0:)');
            WRITE_STRING('NO FAMILIARITY  WITH
THE EXECUTION OF THIS  (:0:)');
            WRITE_STRING('PROGRAM.  IF THIS IS
YOUR FIRST EXECUTION OF  (:0:)');
            WRITE_STRING('THE   PROGRAM,    I
STRONGLY RECOMMEND YOU SELECT  (:0:)');
            WRITE_STRING('THE  FULL  PROMPTING
OPTION.  CONVERSLY, THE   (:0:)');
            WRITE_STRING('OPTION  OF   PARTIAL
PROMPTING IS INTENDED FOR  (:0:)');
            WRITE_STRING('THE  USER   WHO   IS
SOMEWHAT FAMILIAR WITH THE  (:0:)');
            WRITE_STRING('EXECUTION   SEQUENCE
OF THIS PROGRAM.  THERE IS  (:0:)');
            WRITE_STRING('LESS DETAIL PROVIDED
IN THIS PROMPTING OPTION.  (:0:)');
            WRITE_STRING('PRESS THE RETURN KEY
WHEN YOU ARE FINISHED  (:0:)');
            WRITE_STRING('READING  WITH  THIS
PAGE.  (:0:)');
            FOR I := 1 TO 4 DO  DISPLAY(NL);
```

```
            ACCEPT(DUMMY);
            FOR I := 1 TO 25 DO  DISPLAY(NL);
            WRITE_STRING('THE    EXECUTION    OF
THIS PROGRAM HAS BEEN MADE  (:0:)');
            WRITE_STRING('AS    PAINLESS    AS
POSSIBLE.  HOWEVER, I FULLY  (:0:)');
            WRITE_STRING('REALIZE THAT YOU MAY
NOT BE AN EXPERT IN THE  (:0:)');
            WRITE_STRING('THEORY  OF  SOFTWARE
RELIABILITY MODELS AND  (:0:)');
            WRITE_STRING('THAT  YOU  MAY  FEEL
UNCOMFORTABLE WITH SOME OF  (:0:)');
            WRITE_STRING('THE QUESTIONS  POSED
DURING THE EXECUTION  (:0:)');
            WRITE_STRING('SEQUENCE.  FOR  THIS
REASON, YOU WILL BE  (:0:)');
            WRITE_STRING('AFFORDED         THE
OPPORTUNITY TO ASK FOR HELP  (:0:)');
            WRITE_STRING('AT ANY  TIME  DURING
THE INITIAL QUESTIONING  (:0:)');
            WRITE_STRING('PROCESS.  EACH  CALL
FOR HELP WILL BE KEYED  (:0:)');
            WRITE_STRING('TO  THE  PARTICULAR
QUESTION WHICH WAS BEING  (:0:)');
            WRITE_STRING('ASKED AT THE TIME OF
THE CALL FOR HELP.  (:0:)');
            WRITE_STRING('PRESS THE RETURN KEY
WHEN WHEN YOU ARE FINISH-  (:0:)');
            WRITE_STRING('ED    READING    THIS
PAGE.  (:0:)');
            FOR I := 1 TO 8 DO  DISPLAY(NL);
            ACCEPT(DUMMY);
            CALL_FOR_HELP:=0;
            FOR I := 1 TO 25 DO  DISPLAY(NL)
          END; "CALL FOR HELP = 1"


        IF(CALL_FOR_HELP = 2)
          THEN
            BEGIN "CALL FOR HELP = 2"
              WRITE_STRING('THE    LEVEL    OF
PROMPTING REFERS TO THE DEGREE  (:0:)');
              WRITE_STRING('OF     EXPLANATION
PRESENTED IN EACH QUESTION OF  (:0:)');
              WRITE_STRING('THE      EXECUTION
SEQUENCE.  THERE ARE TWO LEVELS  (:0:)');
              WRITE_STRING('OF       PROMPTING
AVAILABLE TO YOU, FULL OR PARTIAL  (:0:)');
              WRITE_STRING('PROMPTING.    FULL
PROMPTING PROVIDES COMPLETE  (:0:)');
              WRITE_STRING('EXPLANATIONS    OF
QUESTIONS AND POSSIBLE  (:0:)');
              WRITE_STRING('RESPONSES.  IT  IS
TO BE ASSOCIATED WITH A  (:0:)');
```

```
              WRITE_STRING('USER  WHO  IS  NOT
FAMILIAR WITH THE EXECUTION  (:0:)');
              WRITE_STRING('SEQUENCE  OF   THE
PROGRAM.  PARTIAL PROMPTING  (:0:)');
              WRITE_STRING('LEVEL     PROVIDES
MINIMAL INFORMATION PERTAINING  (:0:)');
              WRITE_STRING('TO A QUESTION  AND
ITS SET OF RESPONSES.  IT  (:0:)');
              WRITE_STRING('IS  INTENDED   FOR
USE BY A MORE EXPERIENCED  (:0:)');
              WRITE_STRING('USER    OF    THIS
PROGRAM.  SINCE YOU INVOKED  (:0:)');
              WRITE_STRING('HELP   FROM    THE
PROMPTING LEVEL QUESTION,  (:0:)');
              WRITE_STRING('I  RECOMMEND   YOU
SELECT THE FULL PROMPTING  (:0:)');
              WRITE_STRING('OPTION.  PRESS THE
RETURN KEY WHEN YOU  (:0:)');
              WRITE_STRING('ARE      FINISHED
READING THIS PAGE.  (:0:)');
              FOR I := 1 TO 5 DO  DISPLAY(NL);
              ACCEPT(DUMMY);
              CALL_FOR_HELP:=0;
              FOR I := 1 TO 25 DO  DISPLAY(NL)
            END; "CALL FOR HELP = 2"


        IF(CALL_FOR_HELP = 3)
          THEN
            BEGIN "CALL FOR HELP = 3"
              WRITE_STRING('THERE   ARE    TWO
MODELS WHICH MAY BE USED  (:0:)');
              WRITE_STRING('BY  YOU   IN   THE
EXECUTION OF THIS PROGRAM.  (:0:)');
              WRITE_STRING('YOU    WILL     BE
ALLOWED TO USE ANY COMBINATION  (:0:)');
              WRITE_STRING('OF THE TWO MODELS.
THE TWO MODELS ARE  (:0:)');
              WRITE_STRING('THE
SCHICK-WOLVERTON MODEL AND THE  (:0:)');
              WRITE_STRING('JELINSKI-MORANDA
MODEL.  (:0:)');
              DISPLAY(NL);
              WRITE_STRING('A    SEMI-DETAILED
DISCUSSION OF EACH MODEL  (:0:)');
              WRITE_STRING('FOLLOWS.     PRESS
THE RETURN KEY WHEN YOU  (:0:)');
              WRITE_STRING('HAVE      FINISHED
READING THIS PAGE.  (:0:)');
              FOR   I   :=   1   TO   12   DO
DISPLAY(NL);
              ACCEPT(DUMMY);
              FOR   I   :=   1   TO   25   DO
DISPLAY(NL);
```

```
            WRITE_STRING('THE
SCHICK-WOLVERTON MODEL COMPUTES A  (:0:)');
            WRITE_STRING('A      RELIABILITY
ESTIMATE AND AN ESTIMATE  (:0:)');
            WRITE_STRING('OF THE  MEAN  TIME
TO FAILURE OF A SOFTWARE  (:0:)');
            WRITE_STRING('PROJECT,     BASED
UPON A HAZARD FUNCTION AND  (:0:)');
            WRITE_STRING('AND THE  FOLLOWING
THREE ASSUMPTIONS.  (:0:)');
            WRITE_STRING('THE   AMOUNT    OF
DEGUGGING TIME BETWEEN ERROR  (:0:)');
            WRITE_STRING('OCCURENCES  HAS  A
RAYLEIGH DISTRIBUTION.  (:0:)');
            WRITE_STRING('THE ERROR RATE  IS
PROPORTIONAL TO THE  (:0:)');
            WRITE_STRING('NUMBER  OF  ERRORS
REMAINING AND THE TIME  (:0:)');
            WRITE_STRING('SPENT           IN
DEBUGGING.  FINALLY, EACH  (:0:)');
            WRITE_STRING('ERROR   DISCOVERED
IS IMMEDIATELY REMOVED,  (:0:)');
            WRITE_STRING('THUS REDUCING  THE
ERROR TOTAL BY ONE.  (:0:)');
            WRITE_STRING('THE PARAMATERS  OF
THE MODEL ARE AS  (:0:)');
            WRITE_STRING('FOLLOWS.        1.
TOTAL NUMBER OF INITIAL  (:0:)');
            WRITE_STRING('ERRORS-EITHER
CALCULATED OR ESTIMATED  (:0:)');
            WRITE_STRING('BY  SOME   METHOD.
2.  TIME INTERVAL  (:0:)');
            WRITE_STRING('BETWEEN      ERROR
DISCOVERIES.  3.  TOTAL  (:0:)');
            WRITE_STRING('NUMBER   OF   TIME
INTERVALS.  4.  THE  (:0:)');
            WRITE_STRING('CUMULATIVE  NUMBER
OF ERRORS TO THE  (:0:)');
            WRITE_STRING('PRESENT      TIME.
PRESS THE RETURN KEY WHEN  (:0:)');
            WRITE_STRING('YOU HAVE  FINISHED
READING THIS PAGE.  (:0:)');
            ACCEPT(DUMMY);
            FOR   I   :=   1   TO   25   DO
DISPLAY(NL);
            WRITE_STRING('THE    RELIABILITY
MODEL IS OF THE FORM  (:0:)');
            DISPLAY(NL);
            WRITE_STRING('      R(T)       =
EXP(-PHI*(N-(I-1))(T**2)/2)  (:0:)');
            DISPLAY(NL);
            WRITE_STRING('WHERE  PHI  IS   A
CONSTANT OF PROPORTIONALITY,. (:0:)');
            WRITE_STRING('N  IS  THE   TOTAL
```

```
NUMBER OF ERRORS WHICH  (:0:)');
              WRITE_STRING('ARE  ESTIMATED  TO
BE IN THE PROGRAM,  (:0:)');
              WRITE_STRING('I IS A  PARTICULAR
ERROR OCCURENCE,  (:0:)');
              WRITE_STRING('AND T IS THE  TIME
INTERVAL ASSOCIATED  (:0:)');
              WRITE_STRING('WITH THE ITH ERROR
OCCURENCE.  (:0:)');
              WRITE_STRING('PRESS  THE  RETURN
KEY WHEN YOU ARE  (:0:)');
              WRITE_STRING('FINISHED   READING
THIS PAGE.  (:0:)');
              FOR   I   :=   1   TO   10   DO
DISPLAY(NL);
              ACCEPT(DUMMY);
              FOR   I   :=   1   TO   25   DO
DISPLAY(NL);
              WRITE_STRING('THE
JELINSKI-MORANDA RELIABILITY MODEL  (:0:)');
              WRITE_STRING('COMPUTES          A
RELIABILITY ESTIMATE AND A MEAN  (:0:)');
              WRITE_STRING('TIME   TO   FAILUE
ESTIMATE BASED UPON A  (:0:)');
              WRITE_STRING('HAZARD    FUNCTION
AND THE FOLLOWING FOUR  (:0:)');
              WRITE_STRING('ASSUMPTIONS.    1.
THE AMOUNT OF DEBUGGING  (:0:)');
              WRITE_STRING('TIME BETWEEN ERROR
OCCURENCES HAS AN  (:0:)');
              WRITE_STRING('ERROR    OCCURENCE
RATE PROPORTIONAL TO THE  (:0:)');
              WRITE_STRING('TO THE  NUMBER  OF
ERRORS REMAINING.  (:0:)');
              WRITE_STRING('2.   EACH    ERROR
DISCOVERED IS IMMEDIATELY  (:0:)');
              WRITE_STRING('REMOVED,      THUS
DECREASING THE TOTAL ERRORS  (:0:)');
              WRITE_STRING('BY  ONE.   3.  THE
OCCURENCE RATE BETWEEN  (:0:)');
              WRITE_STRING('ERRORS           IS
CONSTANT.  4. ALL ERRORS WHICH  (:0:)');
              WRITE_STRING('REMAIN   IN    THE
PROGRAM AT ANY GIVEN TIME  (:0:)');
              WRITE_STRING('ARE EQUALLY LIKELY
TO OCCUR.  (:0:)');
              WRITE_STRING('THE PARAMETERS  OF
THE MODEL ARE AS  (:0:)');
              WRITE_STRING('FOLLOWS.       THE
TOTAL NUMBER OF INITIAL  (:0:)');
              WRITE_STRING('ERRORS            -
CALCULATED OR ESTIMATED.  THE  (:0:)');
              WRITE_STRING('TIME      INTERVAL
BETWEEN ERROR DISCOVERIES.  (:0:)');
```

```
              WRITE_STRING('THE  TOTAL  NUMBER
OF TIME INTERVALS.  (:0:)');
              WRITE_STRING('THE   NUMBER   OF
ERRORS FOUND TO PRESENT TIME.  (:0:)');
              WRITE_STRING('PRESS  THE  RETURN
KEY WHEN YOU ARE  (:0:)');
              WRITE_STRING('FINISHED   READING
THIS PAGE.  (:0:)');
              ACCEPT(DUMMY);
              FOR   I   :=   1   TO   25   DO
DISPLAY(NL);
              WRITE_STRING('THE    RELIABILITY
FUNCTION HAS THE FOLLOWING FORM.  (:0:)');
              DISPLAY(NL);
              WRITE_STRING('      R(T)       =
EXP(-PHI*(N-N1)*T),  WHERE  (:0:)');
              DISPLAY(NL);
              WRITE_STRING('PHI      IS      A
PROPORTIONALITY CONSTANT, N IS  (:0:)');
              WRITE_STRING('THE  TOTAL  NUMBER
OF INITIAL ERRORS, N1  (:0:)');
              WRITE_STRING('IS  A  PARTICULAR
ERROR OCCURENCE, AND T  (:0:)');
              WRITE_STRING('IS    THE     TIME
INTERVAL ASSOCIATED WITH THE  (:0:)');
              WRITE_STRING('PARTICULAR   ERROR
OCCURENCE.  (:0:)');
              DISPLAY(NL);
              WRITE_STRING('PRESS  THE  RETURN
KEY WHEN YOU ARE  (:0:)');
              WRITE_STRING('FINISHED   READING
THIS PAGE.  (:0:)');
              FOR   I   :=   1   TO   10   DO
DISPLAY(NL);
              ACCEPT(DUMMY);
              CALL_FOR_HELP := 0;
              FOR I := 1 TO 25 DO  DISPLAY(NL)
            END; "CALL FOR HELP = 3"


        IF(CALL_FOR_HELP = 4)
          THEN
            BEGIN "CALL FOR HELP = 4"
              WRITE_STRING('THERE   ARE   TWO
FORMS OF SOLUTIONS WHICH  (:0:)');
              WRITE_STRING('YOU MAY CHOOSE  IN
THIS PROGRAM.  ONE IS  (:0:)');
              WRITE_STRING('A        GRAPHICAL
SOLUTION AND THE OTHER IS A  (:0:)');
              WRITE_STRING('TABULAR  SOLUTION.
THE GRAPHICAL FORM  (:0:)');
              WRITE_STRING('CONTAINS       THE
COMPUTED RELIABILITY AND  (:0:)');
              WRITE_STRING('IS PLOTTED AGAINST
```

```
THE COMPUTED  (:0:)');
              WRITE_STRING('MEAN    TIME    TO
FAILURE.  THERE MAY BE  (:0:)');
              WRITE_STRING('MULTIPLE    MODELS
PRESENTING DATA ON  (:0:)');
              WRITE_STRING('THE  GRAPH.   THIS
GRAPH MAY BE PRESENTED (:0:)');
              WRITE_STRING('AT EITHER  OF  TWO
LOCATIONS.  THESE LOCATIONS (:0:)');
              WRITE_STRING('ARE THE CHROMATICS
COLOR CRT OR THE PLOTTER (:0:)');
              WRITE_STRING('DISPLAY    DEVICE.
THE TABULAR SOLUTION (:0:)');
              WRITE_STRING('IS    MERELY     A
PRESENTATION OF SOLUTIONS  (:0:)');
              WRITE_STRING('COMPUTED  BY   THE
MODEL OR MODELS IN  (:0:)');
              WRITE_STRING('A  READABLE  FORM.
IF YOU ARE NOT  (:0:)');
              WRITE_STRING('FAMILIAR WITH  THE
EXECUTION OF THIS  (:0:)');
              WRITE_STRING('PROGRAM,          I
RECOMMEND BOTH SOLUTIONS.  (:0:)');
              WRITE_STRING('PRESS  THE  RETURN
KEY WHEN YOU ARE  (:0:)');
              WRITE_STRING('FINISHED   READING
THIS PAGE.  (:0:)');
              FOR I := 1 TO 3 DO  DISPLAY(NL);
              ACCEPT(DUMMY);
              CALL_FOR_HELP := 0;
              FOR I := 1 TO 25 DO  DISPLAY(NL)
            END; "CALL FOR HELP = 4)"


        IF(CALL_FOR_HELP = 5)
          THEN
            BEGIN "CALL FOR HELP = 5"
              WRITE_STRING('THE SCALE  OF  THE
GRAPHICAL SOLUTION IS  (:0:)');
              WRITE_STRING('ONLY          PARTLY
DETERMINED BY YOU.  THE  (:0:)');
              WRITE_STRING('SCALE    OF    THE
VERTICAL AXIS, OR THE MEAN  (:0:)');
              WRITE_STRING('TIME  TO   FAILURE
AXIS, IS THE VARIABLE  (:0:)');
              WRITE_STRING('SCALE AXIS AND YOU
MUST ENTER THE SCALE  (:0:)');
              WRITE_STRING('WHICH  YOU  DESIRE
TO SEE PRESENTED.  THE  (:0:)');
              WRITE_STRING('CANDIDATES     ARE
DAYS, WEEEKS, OR MONTHS.  (:0:)');
              WRITE_STRING('SINCE   THE   MEAN
TIME TO FAILURE IS A  (:0:)');
              WRITE_STRING('MEASURE  OF  TIME,
```

```
IT IS OBVIOUS THAT THE  (:0:)');
              WRITE_STRING('SCALE  WOULD  NEED
TO BE IN TIME.  INSURE  (:0:)');
              WRITE_STRING('THAT THE SCALE YOU
SELECT FOR THIS AXIS  (:0:)');
              WRITE_STRING('IS COMPATIBLE WITH
THE DATA YOU HAVE  (:0:)');
              WRITE_STRING('ENTERED  FOR  YOUR
MODEL OR MODELS.  FOR  (:0:)');
              WRITE_STRING('INSTANCE,  IF  YOU
HAVE SELECTED THE  (:0:)');
              WRITE_STRING('SCHICK-WOLVERTON
MODEL AND YOUR TIME  (:0:)');
              WRITE_STRING('INTERVAL OF  ERROR
DISCOVERY IS IN DAYS,  (:0:)');
              WRITE_STRING('THEN  YOUR   SCALE
FOR THE MEAN TIME TO  (:0:)');
              WRITE_STRING('FAILURE       AXIS
SHOULD ALSO BE IN DAYS.  (:0:)');
              WRITE_STRING('IF IT IS NOT,  YOU
RUN THE RISK OF BEING  (:0:)');
              WRITE_STRING('PRESENTED  WITH  A
WILDLY SKEWED GRAPH.  (:0:)');
              WRITE_STRING('PRESS  THE  RETURN
KEY WHEN YOU ARE  (:0:)');
              WRITE_STRING('FINISHED   READING
THIS PAGE.  (:0:)');
              ACCEPT(DUMMY);
              CALL_FOR_HELP := 0;
              FOR I := 1 TO 25 DO  DISPLAY(NL)
            END; "CALL FOR HELP = 5"


        IF(CALL_FOR_HELP = 6)
          THEN
            BEGIN "CALL FOR HELP = 6"
              WRITE_STRING('DATA TO BE ENTERED
FOR THE MODELS  (:0:)');
              WRITE_STRING(' CAN BE DIVIDED OR
CHARACTERIZED INTO  (:0:)');
              WRITE_STRING('FOUR          MAIN
CATEGORIES.  THESE ARE  (:0:)');
              WRITE_STRING('THE  TOTAL  NUMBER
OF ERRORS ESTIMATED  (:0:)');
              WRITE_STRING('TO BE  PRESENT  IN
THE SOFTWARE PACKAGE  (:0:)');
              WRITE_STRING('YOU           ARE
CONSIDERING, THE TIME INTERVAL  (:0:)');
              WRITE_STRING('ASSOCIATED    WITH
ERROR DISCOVERIES, THE  (:0:)');
              WRITE_STRING('NUMBER          OF
INTERVALS PRESENT, AND THE  (:0:)');
              WRITE_STRING('CUMULATIVE  NUMBER
OF ERRORS DISCOVERED  (:0:)');
```

```
              WRITE_STRING('TO   THE   PRESENT
TIME.  YOU MUST INSURE  (:0:)');
              WRITE_STRING('THAT  THESE  DATA
VALUES ARE POSITIVE  (:0:)');
              WRITE_STRING('INTEGER   VALUES.
IF NOT, YOUR INPUT  (:0:)');
              WRITE_STRING('WILL BE IDENTIFIED
IN ERROR AND YOU  (:0:)');
              WRITE_STRING('WILL    HAVE    TO
RE-ENTER THE DATA.  (:0:)');
              WRITE_STRING('PRESS  THE  RETURN
KEY WHEN YOU ARE  (:0:)');
              WRITE_STRING('FINISHED   READING
THIS PAGE.  (:0:)');
              FOR I := 1 TO 6 DO  DISPLAY(NL);
              ACCEPT(DUMMY);
              FOR   I   :=   1   TO   25   DO
DISPLAY(NL);
              WRITE_STRING('THE TWO MODELS  OF
THIS PROGRAM WILL  (:0:)');
              WRITE_STRING('ASSUME THAT  THERE
IS ONLY ONE ERROR  (:0:)');
              WRITE_STRING('DISCOVERED IN EACH
INTERVAL.  THEREFORE,  (:0:)');
              WRITE_STRING('THE     CUMULATIVE
ERRORS DISCOVERED IS  (:0:)');
              WRITE_STRING('EQUAL TO THE TOTAL
NUMBER OF INTERVALS.  (:0:)');
              WRITE_STRING('ADDITIONALLY,   DO
NOT ALLOW THE TOTAL  (:0:)');
              WRITE_STRING('INTERVALS TO EQUAL
OR EXCEED THE INITIAL  (:0:)');
              WRITE_STRING('ERRORS.   YOU  ARE
RESTRICTED TO A MAXIMUM  (:0:)');
              WRITE_STRING('OF    250    TOTAL
INTERVALS.  FINALLY, IF YOU  (:0:)');
              WRITE_STRING('SELECTED BOTH  THE
MODELS FOR SOLUTIONS,  (:0:)');
              WRITE_STRING('YOU ONLY  NEED  TO
ENTER THE DATA ONE TIME.  (:0:)');
              WRITE_STRING('PRESS  THE  RETURN
KEY WHEN YOU  (:0:)');
              WRITE_STRING('ARE      FINISHED
READING THIS PAGE.  (:0:)');
              FOR I := 1 TO 9 DO  DISPLAY(NL);
              ACCEPT(DUMMY);
              FOR   I   :=   1   TO   25   DO
DISPLAY(NL);
              CALL_FOR_HELP:=0
            END; "CALL FOR HELP = 6"


          IF(CALL_FOR_HELP = 7)
            THEN
```

```
BEGIN "CALL FOR HELP = 7"
    WRITE_STRING('YOU   WILL    BE
ALLOWED TO DISPLAY A GRAPHICAL (:0:)');
    WRITE_STRING('SOLUTION ANY  AT
ONE OF TWO DEVICES.  THET (:0:)');
    WRITE_STRING('DEVICES ARE  THE
CHROMATICS COLOR CRT AND THE (:0:)');
    WRITE_STRING('THE     PLOTTER.
YOU MUST PERFORM CERTAIN (:0:)');
    WRITE_STRING('SYSTEM
CONFIGURATION ACTIONS PRIOR TO USING  (:0:)');
    WRITE_STRING('THESE    DEVICES
HOWEVER.  IF YOU HAVE NOT DONE (:0:)');
    WRITE_STRING('THIS  YET,   YOU
CANNOT USE THESE OPTIONS.  SEE (:0:)');
    WRITE_STRING('THE USERS MANUAL
FOR A DESCRIPTION OF ACTIONS(:0:)');
    WRITE_STRING('NECESSARY     TO
CONFIGURE THE SYSTEM TO ENABLE  (:0:)');
    WRITE_STRING('THE INTERDATA TO
COMMUNICATE WITH THESE OTHER(:0:)');
    WRITE_STRING('DEVICES.   PRESS
THE RETURN KEY WHEN YOU ARE  (:0:)');
    WRITE_STRING('FINISHED READING
THIS PAGE.                (:0:)');
    FOR  I   :=   1   TO   10   DO
DISPLAY(NL);
    ACCEPT(DUMMY);
    FOR  I   :=   1   TO   25   DO
DISPLAY(NL);
    CALL_FOR_HELP := 0
END  "CALL FOR HELP = 7"


END; "PROCEDURE HELP"
```

```
   PROCEDURE SQUARE_ROOT(VALUE : REAL);


     "*********************************************************"
     "*
*"
     "* THIS PROCEDURE  CALCULATES  THE SQUARE
ROOT OF A     *"
     "* NUMBER AND RETURNS  IT  TO THE CALLING
PROCEDURE IN *"
     "*  A   VARIABLE   NAMED   ANSWER.    THE
PROCEDURE IS INVOKED*"
     "* FROM  EITHER  OF  THE  TWO RELIABILITY
MODEL COMPUTAT-*"
     "*            ION             PROCEDURES.
*"
     "*
*"
     "*********************************************************"


     CONST
       DELTA          = 0.001;
       EPSILON        = 0.001;

     VAR
       ROOT           : REAL;

       BEGIN "PROCEDURE SQUARE_ROOT"
         ROOT          := 0.0;
         ANSWER        := 0.0;
         IF(VALUE < DELTA)
           THEN
             ANSWER := 0.0
           ELSE
             BEGIN
               ROOT := 1.0;
               REPEAT
                 ROOT   :=    (VALUE/ROOT    +
ROOT)/2.0
               UNTIL ABS(VALUE/(ROOT * ROOT) -
1.0) < EPSILON;
               ANSWER := ROOT
             END
       END; "PROCEDURE SQUARE_ROOT"
```

```
   PROCEDURE
COLLECT_GRAPH_INFORMATION(S_W_SELECTED,J_M_SELECTED,ALL_SELE
: BOOLEAN);


      "#########################################################
      "#
#"
      "# THIS PROCEDURE GATHERS THE  NECESSARY
INFORMATION FROM THE#"
      "#  USER  TO  DETERMINE  WHICH   DISPLAY
DEVICE HE/SHE WISHES THE #"
      "# GRAPHICAL SOLUTION  TO  BE PRESENTED.
THE PROCEDURE IS     #"
      "# INVOKED  FROM  BOTH  OF THE PROMPTING
PROCEDURES.          #"
      "#
#"
      "#########################################################


     CONST
       CHROMATICS              = 'C';
       SPINWRITER              = 'S';
       PLOTTER                 = 'P';
       CHROM_AND_SPIN          = 'X';
       CHROM_AND_PLOT          = 'B';
       SPIN_AND_PLOT           = 'Z';
       ALL_THREE               = 'A';
       NEED_HELP               = 'H';
       BLANK                   = ' ';

     VAR
       MESSAGE_IN              : CHAR;
       DUMMY                   : CHAR;
       HELP_FLAG               : INTEGER;
       I                       : INTEGER;
       SUCCESSFUL_INPUT        : BOOLEAN;

       BEGIN    "PROCEDURE    COLLECT    GRAPH
INFORMATION"
         MESSAGE_IN            := BLANK;
         DUMMY                 := BLANK;
         HELP_FLAG             := 0;
         I                     := 0;
         SUCCESSFUL_INPUT      := FALSE;
         FOR I := 1 TO 25 DO  DISPLAY(NL);
         WRITE_STRING('YOU MAY  SELECT  EITHER
OF TWO DISPLAY DEVICES (:0:)');
         WRITE_STRING('TO     DEPICT      YOUR
GRAPHICAL SOLUTION ON.  THESE ARE(:0:)');
         WRITE_STRING('THE  CHROMATICS   COLOR
CRT AND THE PLOTTER DEVICE.(:0:)');
```

```
          WRITE_STRING('TO      SELECT      THE
  CHROMATICS COLOR CRT DEVICE, YOU (:0:)');
          WRITE_STRING('MUST   ENTER   C.    TO
  SELECT THE PLOTTER, ENTER P.(:0:)');
          WRITE_STRING('TO  INVOKE   THE   HELP
  PROCEDURE, ENTER AN H.             (:0:)');
          FOR I := 1 TO 15 DO DISPLAY(NL);
          SUCCESSFUL_INPUT := FALSE;
          WHILE(NOT SUCCESSFUL_INPUT)DO
            BEGIN "WHILE"
              WRITE_STRING('PLEASE  ENTER  YOUR
  CHOICE NOW. (:0:)');
              ACCEPT(MESSAGE_IN);
  ACCEPT(DUMMY);
              CASE MESSAGE_IN OF
                CHROMATICS        : BEGIN
                                      SUCCESSFUL_INPUT
  := TRUE;
                                      IF(S_W_SELECTED)
                                        THEN
                                          DESTINATION
  := 1;
                                      IF(J_M_SELECTED)
                                        THEN
                                          DESTINATION
  := 2;
                                      IF(ALL_SELECTED)
                                        THEN
                                          DESTINATION
  := 3
                                    END;
                SPINWRITER        : BEGIN
                                      SUCCESSFUL_INPUT
  := TRUE;
                                      IF(S_W_SELECTED)
                                        THEN
                                          DESTINATION
  := 4;
                                      IF(J_M_SELECTED)
                                        THEN
                                          DESTINATION
  := 5;
                                      IF(ALL_SELECTED)
                                        THEN
                                          DESTINATION
  := 6
                                    END;
                PLOTTER           : BEGIN
                                      SUCCESSFUL_INPUT
  := TRUE;
                                      IF(S_W_SELECTED)
                                        THEN
                                          DESTINATION
  := 7;
```

```
                                             IF(J_M_SELECTED)
                                               THEN
                                                 DESTINATION
  := 8;
                                             IF(ALL_SELECTED)
                                               THEN
                                                 DESTINATION
  := 9
                                           END;
                 CHROM_AND_pLOT        : BEGIN
                                             SUCCESSFUL_INPUT
  := TRUE;
                                             IF(S_W_SELECTED)
                                               THEN
                                                 DESTINATION
  := 10;
                                             IF(J_M_SELECTED)
                                               THEN
                                                 DESTINATION
  := 11;
                                             IF(ALL_SELECTED)
                                               THEN
                                                 DESTINATION
  := 12
                                           END;
                 CHROM_AND_SPIN        : BEGIN
                                             SUCCESSFUL_INPUT
  := TRUE;
                                             IF(S_W_SELECTED)
                                               THEN
                                                 DESTINATION
  := 13;
                                             IF(J_M_SELECTED)
                                               THEN
                                                 DESTINATION
  := 14;
                                             IF(ALL_SELECTED)
                                               THEN
                                                 DESTINATION
  := 15;
                                           END;
                 SPIN_AND_PLOT         : BEGIN
                                             SUCCESSFUL_INPUT
  := TRUE;
                                             IF(S_W_SELECTED)
                                               THEN
                                                 DESTINATION
  := 16;
                                             IF(J_M_SELECTED)
                                               THEN
                                                 DESTINATION
  := 17;
                                             IF(ALL_SELECTED)
                                               THEN
```

```
                                                    DESTINATION
  := 18
                                               END;
                  ALL_THREE            : BEGIN
                                                 SUCCESSFUL_INPUT
  := TRUE;
                                                 IF(S_W_SELECTED)
                                                   THEN
                                                     DESTINATION
  := 19;
                                                 IF(J_M_SELECTED)
                                                   THEN
                                                     DESTINATION
  := 20;
                                                 IF(ALL_SELECTED)
                                                   THEN
                                                     DESTINATION
  := 21
                                               END;
                  NEED_HELP            : BEGIN
                                                 SUCCESSFUL_INPUT
  := FALSE;
                                                 HELP_FLAG
  := 7;
                                                 HELP(HELP_FLAG)
                                               END;
                  ELSE                 : BEGIN
                                                 SUCCESSFUL_INPUT
  := FALSE;
                                                 WRITE_STRING('ERROR
  IN YOUR INPUT.(:0:)')
                                               END
                END "CASE OF MESSAGE IN"
              END; "WHILE"
           FOR I := 1 TO 25 DO  DISPLAY(NL);
           WRITE_STRING('AT THIS TIME PRESS  THE
  BREAK KEY AND(:0:)');
           WRITE_STRING('ENTER THE WORD  PROMPT.
  THIS STEP IS(:0:)');
           WRITE_STRING('NECESSARY   TO   AVOID
  SKEWING THE SCALES(:0:)');
           WRITE_STRING('OF  THE  GRAPH  ON  THE
  DEVICES SELECTED. (:0:)');
           WRITE_STRING('WHEN   YOU   TAKE   THE
  ACTION, YOU WILL  (:0:)');
           WRITE_STRING('NO LONGER  RECEIVE  THE
  PROMPT SYMBOL (:0:)');
           WRITE_STRING('ON THE CRT FACE.   LOOK
  FOR THE CURSOR(:0:)');
           WRITE_STRING('TO   BLINK   WHEN   THE
  PROGRAM IS WAITING(:0:)');
           WRITE_STRING('FOR  INPUT  FROM  YOU.
  THIS SHOULD NOT(:0:)');
           WRITE_STRING('PRESENT A  BIG  PROBLEM
```

```
  AS THE ONLY (:0:)');
          WRITE_STRING('TIME YOU NEED TO  INPUT
  AN ACTION WITH(:0:)');
          WRITE_STRING('THE PROMPT OFF IS AFTER
  THE GRAPH IS(:0:)');
          WRITE_STRING('DRAWN AND  THE  PROGRAM
  HAS HALTED.  AT(:0:)');
          WRITE_STRING('THAT TIME, REENTER  THE
  WORD PROMPT.(:0:)');
          WRITE_STRING('IF  YOU  HAVE  SELECTED
  THE PLOTTER, INSURE(:0:)');
          WRITE_STRING('THAT  DEVICE  PA42:  IS
  NOT ASSIGNED TO(:0:)');
          WRITE_STRING('THE INTERDATA.  IF  YOU
  DO NOT KNOW(:0:)');
          WRITE_STRING('HOW TO  CHECK,  CONSULT
  WITH AN OPERATOR.(:0:)');
          WRITE_STRING('PRESS  THE  RETURN  KEY
  WHEN YOU HAVE (:0:)');
          WRITE_STRING('COMPLETED         THESE
  ACTIONS.       (:0:)');
          FOR I := 1 TO 2 DO  DISPLAY(NL);
          ACCEPT(DUMMY)
        END;    "PROCEDURE    COLLECT    GRAPH
  INFORMATION"
```

```
 PROCEDURE  LOAD_S_W_GRAPH_DATA(SCALE_SELECTED
: INTEGER);

    "*************************************************************
    "*
*"
    "* THIS  PROCEDURE  LOADS  THE  COORDINATE
RECORD INSTANCE    *"
    "* WITH  THE  DATA  NECESSARY  TO PLOT THE
GRAPH ON EITHER OF *"
    "* THE DISPLAY DEVICES.  IT IS CALLED FROM
EITHER OF THE  *"
    "*   TWO   PROMPTING   PROCEDURES.    IT'S
FUNCTIONING IS EXACTLY *"
    "* LIKE THE LOAD_J_M_GRAPH_DATA PROCEDURE,
EXCEPT THAT IT *"
    "*  IS  FOR  THE  SCHICK-WOLVERTON  MODEL.
*"
    "*
*"
    "*************************************************************


   CONST
     CHROMATICS_ORIGIN            = 100;
     CHROMATICS_PIXELS_PER_UNIT = 30;
     PLOTTER_ORIGIN               = 800;
     PLOTTER_PIXELS_PER_UNIT    = 200;

   VAR
     TEMP                          : REAL;
     J                             : INTEGER;

     BEGIN  "PROCEDURE  LOAD  SCHICK-WOLVERTON
GRAPH DATA"
       FOR J := 1 TO ERRORS_DISCOVERED DO
         BEGIN "FOR LOOP"

           "LOAD  RELIABILITY  FOR  CHROMATICS
GRAPH"

             TEMP                            :=
CONV(CHROM_DATA.S_W_PLOT(.J,1.));
             TEMP := TEMP / 10.0;
             TEMP       :=       TEMP          *
CONV(CHROMATICS_PIXELS_PER_UNIT)             +
CONV(CHROMATICS_ORIGIN);
             CHROM_DATA.S_W_PLOT(.J,1.)      :=
ROUND(TEMP);

           "LOAD  MEAN  TIME  TO  FAILURE  FOR
CHROMATICS GRAPH"
```

```
              CHROM_DATA.S_W_PLOT(.J,2.)      :=
CHROM_DATA.S_W_PLOT(.J,2.)                     *
(CHROMATICS_PIXELS_PER_UNIT    DIV    2)      +
CHROMATICS_ORIGIN;
              IF(CHROM_DATA.S_W_PLOT(.J,2.)   >
400)
               THEN
                 CHROM_DATA.S_W_PLOT(.J,2.)  :=
450;

            "LOAD   RELIABILITY   FOR   PLOTTER
GRAPH"

              TEMP                            :=
CONV(PLOT_DATA.S_W_PLOT(.J,1.));
              TEMP := TEMP / 10.0;
              TEMP       :=       TEMP         *
CONV(PLOTTER_PIXELS_PER_UNIT)                  +
CONV(PLOTTER_ORIGIN);
              PLOT_DATA.S_W_PLOT(.J,1.)       :=
ROUND(TEMP);

            "LOAD  MEAN  TIME  TO  FAILURE  FOR
PLOTTER GRAPH"

              PLOT_DATA.S_W_PLOT(.J,2.)       :=
PLOT_DATA.S_W_PLOT(.J,2.)                      *
PLOTTER_PIXELS_PER_UNIT      DIV      4        +
PLOTTER_ORIGIN;
              IF(PLOT_DATA.S_W_PLOT(.J,2.)    >
1900)
               THEN
                 PLOT_DATA.S_W_PLOT(.J,2.)   :=
1900
          END; "FOR LOOP"

       CHROM_DATA.NUMBER_OF_ERRORS_OBSERVED :=
ERRORS_DISCOVERED;
       PLOT_DATA.NUMBER_OF_ERRORS_OBSERVED  :=
ERRORS_DISCOVERED;
       CHROM_DATA.SCALE_DESIRED             :=
SCALE_SELECTED;
       PLOT_DATA.SCALE_DESIRED              :=
SCALE_SELECTED;
       CHROM_DATA.COMBINATION_SELECTED      :=
DESTINATION;
       PLOT_DATA.COMBINATION_SELECTED       :=
DESTINATION
     END;  "PROCEDURE  LOAD  SCHICK-WOLVERTON
GRAPH DATA"
```

```
 PROCEDURE  LOAD_J_M_GRAPH_DATA(SCALE_SELECTED
: INTEGER);


    "*************************************************************
    "*
*"
    "* THIS  PROCEDURE  FUNCTIONS  EXACTLY THE
SAME WAY AND HAS *"
    "*  THE  SAME   OVERALL   PURPOSE  AS  THE
LOAD_S_W_GRAPH_DATA   *"
    "*  PROCEDURE,  EXCEPT   IT   IS  FOR  THE
JELINSKI-MORANDA      *"
    "*                                  MODEL.
*"
    "*
*"
    "*************************************************************


   CONST
     CHROMATICS_ORIGIN            = 100;
     CHROMATICS_PIXELS_PER_UNIT = 30;
     PLOTTER_ORIGIN               = 800;
     PLOTTER_PIXELS_PER_UNIT      = 200;

   VAR
     TEMP                         : REAL;
     J                            : INTEGER;

     BEGIN  "PROCEDURE  LOAD  JELINSKI-MORANDA
GRAPH DATA"
       FOR J := 1 TO ERRORS_DISCOVERED DO
         BEGIN "FOR LOOP"

           "LOAD  RELIABILITY  FOR  CHROMATICS
GRAPH"

             TEMP                             :=
CONV(CHROM_DATA.J_M_PLOT(.J,1.));
             TEMP := TEMP / 10.0;
             TEMP       :=       TEMP         *
CONV(CHROMATICS_PIXELS_PER_UNIT)             +
CONV(CHROMATICS_ORIGIN);
             CHROM_DATA.J_M_PLOT(.J,1.)       :=
ROUND(TEMP);

           "LOAD  MEAN  TIME  TO  FAILURE  FOR
CHROMATICS GRAPH"

             CHROM_DATA.J_M_PLOT(.J,2.)       :=
CHROM_DATA.J_M_PLOT(.J,2.)                   *
(CHROMATICS_PIXELS_PER_UNIT    DIV    2)     +
CHROMATICS_ORIGIN;
             IF(CHROM_DATA.J_M_PLOT(.J,2.)    >
400)
```

```
            THEN
              CHROM_DATA.J_M_PLOT(.J,2.)  :=
450;

          "LOAD  RELIABILITY  FOR  PLOTTER
GRAPH"

           TEMP                          :=
CONV(PLOT_DATA.J_M_PLOT(.J,1.));
           TEMP := TEMP / 10.0;
           TEMP     :=      TEMP          *
CONV(PLOTTER_PIXELS_PER_UNIT)             +
CONV(PLOTTER_ORIGIN);
           PLOT_DATA.J_M_PLOT(.J,1.)     :=
ROUND(TEMP);

          "LOAD  MEAN  TIME  TO  FAILURE  FOR
PLOTTER GRAPH"

           PLOT_DATA.J_M_PLOT(.J,2.)     :=
PLOT_DATA.J_M_PLOT(.J,2.)                 *
PLOTTER_PIXELS_PER_UNIT     DIV     4     +
PLOTTER_ORIGIN;
           IF(PLOT_DATA.J_M_PLOT(.J,2.)   >
1900)
            THEN
              PLOT_DATA.J_M_PLOT(.J,2.)  :=
1900
        END; "FOR LOOP"

      CHROM_DATA.NUMBER_OF_ERRORS_OBSERVED :=
ERRORS_DISCOVERED;
      PLOT_DATA.NUMBER_OF_ERRORS_OBSERVED  :=
ERRORS_DISCOVERED;
      CHROM_DATA.SCALE_DESIRED             :=
SCALE_SELECTED;
      PLOT_DATA.SCALE_DESIRED              :=
SCALE_SELECTED;
      CHROM_DATA.COMBINATION_SELECTED      :=
DESTINATION;
      PLOT_DATA.COMBINATION_SELECTED       :=
DESTINATION
    END;  "PROCEDURE  LOAD  JELINSKI-MORANDA
GRAPH DATA"
```

```
PROCEDURE DRAW_GRAPH;

    "*********************************************************
    "*
*"
    "*    THIS    PROCEDURE    CONTROLS    THE
COORDINATION OF THE EXTERNAL*"
    "* PROGRAMS USED IN DRAWING GRAPHS AT  THE
DISPLAY DEVICES. *"
    "* CONTROL IS EXERCISED USING THE VARIABLE
DESTINATION AS  *"
    "*   THE    END    POINT    FOR    THE    GRAPH.
*"
    "*
*"
    "* CODE   IS   PRESENT   TO   ALLOW   THE   SPIN
WRITER DEVICE TO BE     *"
    "* INVOKED  AT  A  LATER  TIME,  WHEN  THE
PROGRAM IS MODIFIED TO *"
    "*   DRAW    GRAPHS    ON    THE    DEVICE.
*"
    "*
*"
    "*********************************************************

     BEGIN "PROCEDURE DRAW GRAPH"
       CASE DESTINATION OF
         1,2,3:      CHROFIX(CHROM_DATA);
         4,5,6:      SPINFIX(SPIN_DATA);
         7,8,9:      PLOTFIX(PLOT_DATA);
         10,11,12:   BEGIN
                       CHROFIX(CHROM_DATA);
                       PLOTFIX(PLOT_DATA)
                     END;
         13,14,15:   BEGIN
                       CHROFIX(CHROM_DATA);
                       SPINFIX(SPIN_DATA)
                     END;
         16,17,18:   BEGIN
                       SPINFIX(SPIN_DATA);
                       PLOTFIX(PLOT_DATA)
                     END;
         19,20,21:   BEGIN
                       CHROFIX(CHROM_DATA);
                       PLOTFIX(PLOT_DATA);
                       SPINFIX(SPIN_DATA)
                     END
       END "CASE OF DESTINATION"
     END;"PROCEDURE DRAW GRAPH"
```

```
   PROCEDURE LOAD_TABULAR_DISPLAY_S_W;

     "#######################################################
     "#
#"
     "# THIS PROCEDURE PERFORMS THE LOADING OF
THE DATA COM-#"
     "# PUTED  BY  THE  SCHICK-WOLVERTON MODEL
FOR DISPLAY ON  #"
     "# THE TABULAR SOLUTION.  VARIABLE VALUES
ARE CONVERTED#"
     "# FROM REAL VALUES TO INTEGER VALUES FOR
DISPLAY ON   #"
     "#      THIS       SOLUTION        FORM.
#"
     "#
#"
     "#######################################################


       BEGIN "LOAD  TABULAR  DISPLAY  FOR  S_W
MODEL"
         TAB_S_W_RELIABILITY                 :=
ROUND(S_W_RELIABILITY * 100.0);
         TAB_S_W_MTTF := ROUND(S_W_MTTF);
         TAB_S_W_STD                         :=
ROUND(S_W_STANDARD_DEVIATION);
         TAB_S_W_PHI   :=   ROUND(S_W_PHI    *
10000.0);
         TAB_S_W_TIME_TO_FIND_ALL_ERRORS    :=
ROUND(S_W_TIME_TO_FIND_ALL_ERRORS)
       END;  "LOAD  TABULAR  DISPLAY  FOR  S_W
MODEL"
```

```
   PROCEDURE LOAD_TABULAR_DISPLAY_J_M;

     "#########################################################
     "#
#"
     "#  THIS  PROCEDURE  PERFORMS  THE   SAME
FUNCTION AND IN THE   #"
     "#  SAME  MANNER  AS  THE  LOAD   TABULAR
DISPLAY S_W PROCEDURE, #"
     "# EXCEPT THAT IS OBVIOUSLY FOR THE  DATA
OF THE JELINSKI- #"
     "#           MORANDA             MODEL.
#"
     "#
#"
     "#########################################################


       BEGIN "LOAD  TABULAR  DISPLAY  FOR  J_M
MODEL"
         TAB_J_M_RELIABILITY                :=
ROUND(J_M_RELIABILITY * 100.0);
         TAB_J_M_MTTF := ROUND(J_M_MTTF);
         TAB_J_M_STD                        :=
ROUND(J_M_STANDARD_DEVIATION);
         TAB_J_M_pHI   :=   ROUND(J_M_PHI    *
10000.0);
         TAB_J_M_TIME_TO_FIND_ALL_ERRORS    :=
ROUND(J_M_TIME_TO_FIND_ALL_ERRORS)
       END;  "LOAD  TABULAR  DISPLAY  FOR  J_M
MODEL"
```

```
  PROCEDURE
PRINT_TABULAR_DISPLAY(S_W_SELECTED,J_M_SELECTED,ALL_SELECTED
: BOOLEAN);


    "*********************************************************
    "*
*"
    "* THIS PROCEDURE PRINTS THE FRAME WORK OF
THE TABULAR FORM    *"
    "* OF SOLUTION  AND  THEN  PRINTS THE DATA
LOADED BY EACH OF THE *"
    "*   RESPECTIVE   LOAD   TABULAR   DISPLAY
PROCEDURES FOR THEIR RE-    *"
    "*           SPECTIVE              MODELS.
*"
    "*
*"
    "*********************************************************


  VAR
    I     : INTEGER;
    DUMMY : CHAR;

    BEGIN   "PROCEDURE   PRINT   THE   TABULAR
SOLUTION"
      FOR I := 1 TO 25 DO
        DISPLAY(NL);

      IF(S_W_SELECTED)
        THEN
          BEGIN
            WRITE_TABULAR_SOLUTION('TABULAR
SOLUTION (:0:)');
            DISPLAY(NL);
            DISPLAY(NL);
            DISPLAY(NL);
            WRITE_TABULAR_SOLUTION('SCHICK-WOLVERTON
DATA IS AS FOLLOWS. (:0:)');
            DISPLAY(NL);
            DISPLAY(NL);
            IF(TAB_S_W_RELIABILITY < 10)
              THEN
                WRITE_TABULAR_SOLUTION('RELIABILITY
IS 0.0(:0:)')
              ELSE
                WRITE_TABULAR_SOLUTION('RELIABILITY
IS 0.(:0:)');
            WRITE_INTEGER(TAB_S_W_RELIABILITY);
            WRITE_TABULAR_SOLUTION('STANDARD
DEVIATION IS (:0:)');
            WRITE_INTEGER(TAB_S_W_STD);
```

```
            IF(TAB_S_W_PHI < 10)
              THEN
                WRITE_TABULAR_SOLUTION('CONSTANT
OF PROPORTINALITY IS 0.000(:0:)');
            IF((TAB_S_W_PHI   >=    10)    AND
(TAB_S_W_PHI < 100))
              THEN
                WRITE_TABULAR_SOLUTION('CONSTANT
OF PROPORTIONALITY IS 0.00(:0:)');
            IF((TAB_S_W_pHI   >=   100)    AND
(TAB_S_W_PHI < 1000))
              THEN
                WRITE_TABULAR_SOLUTION('CONSTANT
OF PROPORTIONALITY IS 0.0(:0:)');
            IF(TAB_S_W_PHI >= 1000)
              THEN
                WRITE_TABULAR_SOLUTION('CONSTANT
OF PROPORTIONALITY IS 0.(:0:)');
            WRITE_INTEGER(TAB_S_W_PHI);
            WRITE_TABULAR_SOLUTION('MEAN  TIME
TO FAILURE IS (:0:)');
            WRITE_INTEGER(TAB_S_W_MTTF);
            WRITE_TABULAR_SOLUTION('TIME    TO
DISCOVER ALL ERRORS IS (:0:)');
            WRITE_INTEGER(TAB_S_W_TIME_TO_FIND_ALL_ERRORS);
            FOR I := 1 TO 8 DO  DISPLAY(NL)
          END;

      IF(J_M_SELECTED)
        THEN
          BEGIN
            WRITE_TABULAR_SOLUTION('TABULAR
SOLUTION (:0:)');
            DISPLAY(NL);
            DISPLAY(NL);
            DISPLAY(NL);
            WRITE_TABULAR_SOLUTION('JELINSKI-MORANDA
DATA IS AS FOLLOWS. (:0:)');
            DISPLAY(NL);
            DISPLAY(NL);
            IF(TAB_J_M_RELIABILITY < 10)
              THEN
                WRITE_TABULAR_SOLUTION('RELIABILITY
IS 0.0(:0:)')
              ELSE
                WRITE_TABULAR_SOLUTION('RELIABILITY
IS 0.(:0:)');
            WRITE_INTEGER(TAB_J_M_RELIABILITY);
            WRITE_TABULAR_SOLUTION('STANDARD
DEVIATION IS (:0:)');
            WRITE_INTEGER(TAB_J_M_STD);
            IF(TAB_J_M_PHI < 10)
              THEN
                WRITE_TABULAR_SOLUTION('CONSTANT
```

```
OF PROPORTIONALITY IS 0.000(:0:)');
            IF((TAB_J_M_PHI   >=    10)    AND
(TAB_J_M_PHI < 100))
              THEN
                WRITE_TABULAR_SOLUTION('CONSTANT
OF PROPORTIONALITY IS 0.00(:0:)');
            IF((TAB_J_M_PHI   >=   100)    AND
(TAB_J_M_PHI < 1000))
              THEN
                WRITE_TABULAR_SOLUTION('CONSTANT
OF PROPORTIONALITY IS 0.0(:0:)');
            IF(TAB_J_M_PHI >= 1000)
              THEN
                WRITE_TABULAR_SOLUTION('CONSTANT
OF PROPORTIONALITY IS 0.(:0:)');
            WRITE_INTEGER(TAB_J_M_PHI);
            WRITE_TABULAR_SOLUTION('MEAN  TIME
TO FAILURE IS (:0:)');
            WRITE_INTEGER(TAB_J_M_MTTF);
            WRITE_TABULAR_SOLUTION('TIME    TO
DISCOVER ALL ERRORS IS (:0:)');
            WRITE_INTEGER(TAB_J_M_TIME_TO_FIND_ALL_ERRORS);
            FOR I := 1 TO 8 DO  DISPLAY(NL)
          END;

      IF(ALL_SELECTED)
        THEN
          BEGIN
            WRITE_TABULAR_SOLUTION('TABULAR
SOLUTION (:0:)');
            DISPLAY(NL);
            DISPLAY(NL);
            DISPLAY(NL);
            WRITE_TABULAR_SOLUTION('SCHICK-WOLVERTON
DATA IS AS FOLLOWS. (:0:)');
            DISPLAY(NL);
            DISPLAY(NL);
            IF(TAB_S_W_RELIABILITY < 10)
              THEN
                WRITE_TABULAR_SOLUTION('RELIABILITY
IS 0.0(:0:)')
              ELSE
                WRITE_TABULAR_SOLUTION('RELIABILITY
IS 0.(:0:)');
            WRITE_INTEGER(TAB_S_W_RELIABILITY);
            WRITE_TABULAR_SOLUTION('STANDARD
DEVIATION IS (:0:)');
            WRITE_INTEGER(TAB_S_W_STD);
            IF(TAB_S_W_PHI < 10)
              THEN
                WRITE_TABULAR_SOLUTION('CONSTANT
OF PROPORTIONALITY IS 0.000(:0:)');
            IF((TAB_S_W_PHI   >=    10)    AND
(TAB_S_W_PHI < 100))
```

```
              THEN
                WRITE_TABULAR_SOLUTION('CONSTANT
OF PROPORTIONALITY IS 0.00(:0:)');
            IF((TAB_S_W_PHI   >=   100)    AND
(TAB_S_W_PHI < 1000))
              THEN
                WRITE_TABULAR_SOLUTION('CONSTANT
OF PROPORTIONALITY IS 0.0(:0:)');
            IF(TAB_S_W_PHI >= 1000)
              THEN
                WRITE_TABULAR_SOLUTION('CONSTANT
OF PROPORTIONALITY IS 0.(:0:)');
            WRITE_INTEGER(TAB_S_W_PHI);
            WRITE_TABULAR_SOLUTION('MEAN  TIME
TO FAILURE IS (:0:)');
            WRITE_INTEGER(TAB_S_W_MTTF);
            WRITE_TABULAR_SOLUTION('TIME    TO
DISCOVER ALL ERRORS IS (:0:)');
            WRITE_INTEGER(TAB_S_W_TIME_TO_FIND_ALL_ERRORS);
            DISPLAY(NL);
            DISPLAY(NL);
            WRITE_TABULAR_SOLUTION('JELINSKI-MORANDA
DATA IS AS FOLLOWS. (:0:)');
            DISPLAY(NL);
            DISPLAY(NL);
            IF(TAB_J_M_RELIABILITY < 10)
              THEN
                WRITE_TABULAR_SOLUTION('RELIABILITY
IS 0.0(:0:)')
              ELSE
                WRITE_TABULAR_SOLUTION('RELIABILITY
IS 0.(:0:)');
            WRITE_INTEGER(TAB_J_M_RELIABILITY);
            WRITE_TABULAR_SOLUTION('STANDARD
DEVIATION IS (:0:)');
            WRITE_INTEGER(TAB_J_M_STD);
            IF(TAB_J_M_PHI < 10)
              THEN
                WRITE_TABULAR_SOLUTION('CONSTANT
OF PROPORTIONALITY IS 0.000(:0:)');
            IF((TAB_J_M_PHI   >=    10)    AND
(TAB_J_M_PHI < 100))
              THEN
                WRITE_TABULAR_SOLUTION('CONSTANT
OF PROPORTIONALITY IS 0.00(:0:)');
            IF((TAB_J_M_PHI   >=   100)    AND
(TAB_J_M_PHI < 1000))
              THEN
                WRITE_TABULAR_SOLUTION('CONSTANT
OF PROPORTIONALITY IS 0.0(:0:)');
            IF(TAB_J_M_PHI >= 1000)
              THEN
                WRITE_TABULAR_SOLUTION('CONSTANT
OF PROPORTIONALITY IS 0.(:0:)');
```

```
            WRITE_INTEGER(TAB_J_M_PHI);
            WRITE_TABULAR_SOLUTION('MEAN  TIME
TO FAILURE IS (:0:)');
            WRITE_INTEGER(TAB_J_M_MTTF);
            WRITE_TABULAR_SOLUTION('TIME_TO_DISCOVER_ALL_ERR
IS (:0:)');
            WRITE_INTEGER(TAB_J_M_TIME_TO_FIND_ALL_ERRORS)
          END;

      DISPLAY(NL);
      DISPLAY(NL);
      WRITE_TABULAR_SOLUTION('PRESS THE RETURN
KEY WHEN YOU ARE (:0:)');
      WRITE_TABULAR_SOLUTION('FINISHED READING
THIS PAGE.(:0:)');
      DISPLAY(NL);
      ACCEPT(DUMMY)
    END;  "PROCEDURE   PRINT   THE    TABULAR
SOLUTION"
```

```
  PROCEDURE COMPUTE_JELINSKI_MORANDA;

   "#######################################################
   "#
#"
   "#  THIS   PROCEDURE   PERFORMS   ALL   THE
COMPUTATIONS ASSOCIATED#"
   "#  WITH   THE   JELINSKI-MORANDA  SOFTWARE
RELIABILITY MODEL.  #"
   "# COMPUTATIONS PERFOMED  ARE  THE SOFTWARE
RELIABILITY, THE#"
   "# MEAN TIME TO FAILURE, THE CALCULATION OF
THE CONSTANT  #"
   "# OF PROPORTIONALITY,  THE  CALCULATION OF
THE STANDARD    #"
   "#   DEVIATION   ASSOCIATED   WITH    THE
RELIABILITY FUNCTION, AND#"
   "#  THE  ESTIMATED  TIME  TO  DISCOVER  ALL
ERRORS WITHIN THE   #"
   "# SOFTWARE PACKAGE  BEING  EXAMINED.  THIS
PROCEDURE IS    #"
   "# INVOKED FROM EITHER OF THE TWO PROMPTING
PROCEDURES.  #"
   "#
#"
   "#######################################################


    VAR
      PHI_SUB_1                            :
REAL;
      PHI_SUB_2                            :
REAL;
      VALUE                                :
REAL;
      SUM_OF_INTERVALS                     :
ARRAY(.1..MAX_ALLOWED.) OF INTEGER;
      I                                    :
INTEGER;

      BEGIN                       "PROCEDURE
COMPUTE_JELINSKI_MORANDA"
        J_M_TIME_TO_FIND_ALL_ERRORS       :=
0.0;
        J_M_STANDARD_DEVIATION            :=
0.0;
        J_M_PHI                           :=
0.0;
        PHI_SUB_1                         :=
0.0;
        PHI_SUB_2                         :=
0.0;
```

```
        VALUE                                   :=
0.0;
        ANSWER                                  :=
0.0;
        J_M_RELIABILITY                         :=
0.0;
        J_M_MTTF                                :=
0.0;
        I                                   := 0;
        FOR I := 1 TO MAX_ALLOWED DO
          SUM_OF_INTERVALS(.I.)             := 0;
        SUM_OF_INTERVALS(.1.)                   :=
J_M_DATA_1(.1.);
        FOR I := 2 TO ERRORS_DISCOVERED DO
          SUM_OF_INTERVALS(.I.)                 :=
SUM_OF_INTERVALS(.I-1.) + J_M_DATA_1(.I.);
        FOR I := 1 TO ERRORS_DISCOVERED DO
          BEGIN "FOR LOOP"
            PHI_SUB_1 := CONV(INITIAL_ERRORS *
SUM_OF_INTERVALS(.I.));
            PHI_SUB_2:=PHI_SUB_2 +  CONV((I-1)
* J_M_DATA_1(.I.));
            J_M_PHI  :=  CONV(I)/(PHI_SUB_1  -
PHI_SUB_2);
            J_M_MTTF   :=    1.0/(J_M_PHI    *
CONV(INITIAL_ERRORS - I));
            CHROM_DATA.J_M_PLOT(.I,2.)      :=
ROUND(J_M_MTTF);
            PLOT_DATA.J_M_PLOT(.I,2.)       :=
ROUND(J_M_MTTF);
            SPIN_DATA.J_M_PLOT(.I,2.)       :=
ROUND(J_M_MTTF);
            VALUE      :=      -J_M_PHI      *
CONV((INITIAL_ERRORS- I) * J_M_DATA_1(.I.));
            J_M_RELIABILITY := DEXP(VALUE);
            CHROM_DATA.J_M_PLOT(.I,1.)      :=
ROUND(J_M_RELIABILITY * 100.0);
            PLOT_DATA.J_M_PLOT(.I,1.)       :=
ROUND(J_M_RELIABILITY * 100.0);
            SPIN_DATA.J_M_PLOT(.I,1.)       :=
ROUND(J_M_RELIABILITY * 100.0);
          END; "FOR LOOP"
        VALUE := 0.0;
        FOR  I  :=  1  TO  (INITIAL_ERRORS   -
ERRORS_DISCOVERED) DO
          BEGIN "FOR LOOP"
            J_M_TIME_TO_FIND_ALL_ERRORS     :=
(J_M_TIME_TO_FIND_ALL_ERRORS + (1.0/CONV(I)));
            VALUE := VALUE + 1.0/CONV(I * I)
          END; "FOR LOOP"
        J_M_TIME_TO_FIND_ALL_ERRORS         :=
J_M_TIME_TO_FIND_ALL_ERRORS/J_M_PHI;
        IF((VALUE <= 1.1) AND (VALUE >= 0.9))
          THEN
```

```
            ANSWER := 1.0
          ELSE
            SQUARE_ROOT(VALUE);
        J_M_STANDARD_DEVIATION                  :=
 ANSWER/J_M_PHI
      END;                              "PROCEDURE
 COMPUTE_JELINSKI_MORANDA"
```

```
   PROCEDURE COMPUTE_SCHICK_WOLVERTON;

     "*********************************************************
     "*
*"
     "* THIS PROCEDURE PERFORMS THE SAME
FUNCTION AS THE PRIOR *"
     "* PROCEDURE, EXCEPT OF COURSE FOR THE
SCHICK-WOLVERTON *"
     "* SOFTWARE RELIABILITY MODEL AND NOT THE
JELINSKI- *"
     "* MORANDA MODEL. ALTHOUGH THE BASIS OF
THE COMPUTATIONS *"
     "* BETWEEN THE TWO MODELS IS DIFFERENT,
THE VARIOUS FORMS *"
     "* OF ANSWERS BEING SUPPLIED BY THIS MODEL
ARE THE SAME. *"
     "*
*"
     "*********************************************************

     CONST
       PIE                                          =
3.14159265359;

     VAR
       PHI_SUB_1                                    :
REAL;
       PHI_SUB_2                                    :
REAL;
       PHI_SUB_3                                    :
REAL;
       VALUE                                        :
REAL;
       SUM_OF_INTERVALS                             :
INTEGER;
       I                                            :
INTEGER;

      BEGIN                            "PROCEDURE
COMPUTE_SCHICK_WOLVERTON"
        S_W_TIME_TO_FIND_ALL_ERRORS                :=
0.0;
        S_W_STANDARD_DEVIATION                     :=
0.0;
        S_W_PHI                                    :=
0.0;
        PHI_SUB_1                                  :=
0.0;
        PHI_SUB_2                                  :=
```

```
0.0;
        PHI_SUB_3                                :=
0.0;
        VALUE                                    :=
0.0;
        S_W_RELIABILITY                          :=
0.0;
        S_W_MTTF                                 :=
0.0;
        ANSWER                                   :=
0.0;
        SUM_OF_INTERVALS                     := 0;
        I                                    := 0;
        FOR I := 1 TO ERRORS_DISCOVERED DO
          BEGIN "FOR LOOP"
            PHI_SUB_1    :=    PHI_SUB_1     +
2.0/CONV(INITIAL_ERRORS - (I-1));
            PHI_SUB_2    :=    PHI_SUB_2     +
CONV(S_W_DATA_1(.I.)*S_W_DATA_1(.I.));
            PHI_SUB_3 := 1.0/PHI_SUB_2;
            S_W_PHI := PHI_SUB_1 * PHI_SUB_3;
            VALUE      :=      -S_W_PHI      *
CONV((INITIAL_ERRORS      -      (I-1))     *
S_W_DATA_1(.I.) * S_W_DATA_1(.I.)) / 2.0;
            S_W_RELIABILITY := DEXP(VALUE);
            CHROM_DATA.S_W_PLOT(.I,1.)      :=
ROUND(S_W_RELIABILITY * 100.0);
            PLOT_DATA.S_W_PLOT(.I,1.)       :=
ROUND(S_W_RELIABILITY * 100.0);
            SPIN_DATA.S_W_PLOT(.I,1.)       :=
ROUND(S_W_RELIABILITY * 100.0);
            VALUE  :=  PIE/(2.0  *  S_W_PHI  *
CONV(INITIAL_ERRORS - I));
            IF((VALUE <=  1.1)  AND  (VALUE >=
0.9))
              THEN
                ANSWER := 1.0
              ELSE
                SQUARE_ROOT(VALUE);
            S_W_MTTF := ANSWER;
            CHROM_DATA.S_W_PLOT(.I,2.)      :=
ROUND(S_W_MTTF);
            PLOT_DATA.S_W_PLOT(.I,2.)       :=
ROUND(S_W_MTTF);
            SPIN_DATA.S_W_PLOT(.I,2.)       :=
ROUND(S_W_MTTF)
          END; "FOR LOOP"
          VALUE := 0.0;
          FOR I  :=  1  TO  (INITIAL_ERRORS  -
ERRORS_DISCOVERED) DO
            BEGIN "FOR LOOP"
              S_W_TIME_TO_FIND_ALL_ERRORS   :=
S_W_TIME_TO_FIND_ALL_ERRORS + (1.0/CONV(I));
              VALUE := VALUE + 1.0/CONV(I * I)
```

```
            END; "FOR LOOP"
          S_W_TIME_TO_FIND_ALL_ERRORS      :=
 S_W_TIME_TO_FIND_ALL_ERRORS / S_W_PHI;
          IF((VALUE  <=  1.1)  AND  (VALUE  >=
 0.9))
            THEN
              ANSWER := 1.0
            ELSE
              SQUARE_ROOT(VALUE);
          S_W_STANDARD_DEVIATION           :=
 ANSWER/S_W_PHI
       END;                          "PROCEDURE
 COMPUTE_SCHICK_WOLVERTON"
```

```
PROCEDURE PARTIAL_PROMPTING;

"*******************************************************
"*
*"
"* THIS PROCEDURE IS ONE OF THE TWO  MAIN
DRIVERS OF THIS      *"
"* PROGRAM,  THE  OTHER  BEING  PROCEDURE
FULL_PROMPTING.  FROM  *"
"* THIS PROCEDURE, THE USER IS QUERIED AS
TO HIS DATA AND THE*"
"*  DATA  IS  COLLECTED  AND  COORDINATED
THROUGH THE COMPUTATION *"
"*  AND  SOLUTION  PRESENTATION  PROCESS.
THIS PROCEDURE HAS    *"
"* ACCESS TO  ALMOST  ANY OTHER PROCEDURE
WITHIN THIS PROGRAM  *"
"   *   AND   TYPICALLY   CALLS   SEVERAL
PROCEDURES TO COMPUTE, LOAD,  *"
"*    AND    DISPLAY    SOLUTION    FORMS.
*"
"*
*"
"* THIS  PROCEDURE  IS  DESIGNED  FOR THE
MORE EXPERIENCED USER  *"
"*  OF  THE  PROGRAM  AND  PROVIDES  ONLY
MINIMAL DETAIL TO ENABLE *"
"* THE USER  TO  SUCCESSFULLY EXECUTE THE
PROGRAM.            *"
"*
*"
"*******************************************************

CONST
  BLANK                = ' ';
  SCHICK_WOLVERTON     = 'S';
  JELINSKI_MORANDA     = 'J';
  NEED_HELP            = 'H';
  DO_NOT_NEED_HELP     = 'N';
  GRAPHICAL            = 'G';
  TABULAR              = 'T';
  BOTH                 = 'B';
  DAYS                 = 'D';
  WEEKS                = 'W';
  MONTHS               = 'M';

VAR
  SUCCESSFUL_INPUT     : BOOLEAN;
  S_W_SELECTED         : BOOLEAN;
  J_M_SELECTED         : BOOLEAN;
  ALL_SELECTED         : BOOLEAN;
```

```
       HELP_FLAG                : INTEGER;
       NUMBER_IN                : INTEGER;
       SOLUTION_SELECTED        : INTEGER;
       SCALE_SELECTED           : INTEGER;
       ERROR_COUNTER            : INTEGER;
       I                        : INTEGER;
       MESSAGE_IN               : CHAR;
       DUMMY                    : CHAR;


     BEGIN "PROCEDURE PARTIAL PROMPTING"
       SUCCESSFUL_INPUT         := FALSE;
       S_W_SELECTED             := FALSE;
       J_M_SELECTED             := FALSE;
       ALL_SELECTED             := FALSE;
       HELP_FLAG                := 0;
       INITIAL_ERRORS           := 0;
       TOTAL_INTERVALS          := 0;
       ERRORS_DISCOVERED        := 0;
       NUMBER_IN                := 0;
       SOLUTION_SELECTED        := 0;
       SCALE_SELECTED           := 0;
       ERROR_COUNTER            := 0;
       I                        := 0;
       MESSAGE_IN               := BLANK;
       DUMMY                    := BLANK;
       FOR I := 1 TO MAX_ALLOWED DO
         BEGIN
           S_W_DATA_1(.I.)    := 0;
           J_M_DATA_1(.I.)    := 0
         END;
       WRITE_STRING('ENTER THE MODEL OR MODELS
YOU DESIRE  (:0:)');
       WRITE_STRING('TO USE IN THIS EXECUTION.
ENTER S  (:0:)');
       WRITE_STRING('FOR THE  SCHICK-WOLVERTON
MODEL, ENTER  (:0:)');
       WRITE_STRING('J        FOR        THE
JELINSKI-MORANDA MODEL, OR  (:0:)');
       WRITE_STRING('B FOR  A  COMBINATION  OF
BOTH THE MODELS.  (:0:)');
       WRITE_STRING('PLEASE ENTER YOUR  CHOICE
NOW.  (:0:)');
       FOR I := 1 TO 16 DO  DISPLAY(NL);
        SUCCESSFUL_INPUT:=FALSE;
        WHILE(NOT SUCCESSFUL_INPUT)DO
          BEGIN "WHILE"
            ACCEPT(MESSAGE_IN); ACCEPT(DUMMY);
            CASE MESSAGE_IN OF
              SCHICK_WOLVERTON : BEGIN
                                   SUCCESSFUL_INPUT:=TRUE;
                                   S_W_SELECTED:=TRUE
                                 END;
              JELINSKI_MORANDA : BEGIN
```

```
                                    SUCCESSFUL_INPUT:=TRUE;
                                    J_M_SELECTED:=TRUE
                                  END;
            BOTH                : BEGIN
                                    SUCCESSFUL_INPUT:=TRUE;
                                    ALL_SELECTED:=TRUE
                                  END;
            NEED_HELP           : BEGIN
                                    SUCCESSFUL_INPUT:=FALSE;
                                    HELP_FLAG:=3;
                                    HELP(HELP_FLAG);
                                    WRITE_STRING('ENTER
YOUR MODEL CHOICE.(:0:)')
                                  END;
            ELSE                : BEGIN
                                    SUCCESSFUL_INPUT:=FALSE;
                                    WRITE_STRING('ERROR
IN YOUR INPUT. (:0:)')
                                  END
          END "CASE OF MESSAGE IN"
        END; "WHILE"
       FOR I := 1 TO 25 DO  DISPLAY(NL);;
       WRITE_STRING('WHICH TYPE OF SOLUTION DO
YOU DESIRE?  (:0:)');
       WRITE_STRING('ENTER G FOR THE GRAPHICAL
SOLUTION, T  (:0:)');
       WRITE_STRING('FOR THE TABULAR SOLUTION,
OR B FOR  (:0:)');
       WRITE_STRING('BOTH SOLUTION FORMS.   IF
YOU CHOOSE THE (:0:)');
       WRITE_STRING('GRAPHICAL  SOLUTION,  YOU
WILL BE QUERIED  (:0:)');
       WRITE_STRING('AS  TO  YOUR  CHOICE   OF
DISPLAY DEVICES  (:0:)');
       WRITE_STRING('AT A LATER TIME.   PLEASE
ENTER YOUR  (:0:)');
       WRITE_STRING('ANSWER  AT   THIS   TIME.
(:0:)');
       FOR I := 1 TO 14 DO  DISPLAY(NL);
       SUCCESSFUL_INPUT := FALSE;
       WHILE(NOT SUCCESSFUL_INPUT)DO
         BEGIN "WHILE"
           ACCEPT(MESSAGE_IN); ACCEPT(DUMMY);
           CASE MESSAGE_IN OF
             GRAPHICAL : BEGIN
                           SUCCESSFUL_INPUT:=TRUE;
                           SOLUTION_SELECTED:=1
                         END;
             TABULAR   : BEGIN
                           SUCCESSFUL_INPUT:=TRUE;
                           SOLUTION_SELECTED:=2
                         END;
             BOTH      : BEGIN
                           SUCCESSFUL_INPUT:=TRUE;
```

```
                              SOLUTION_SELECTED:=3
                            END;
            NEED_HELP : BEGIN
                              SUCCESSFUL_INPUT:=FALSE;
                              HELP_FLAG:=4;
                              HELP(HELP_FLAG);
                              WRITE_STRING('ENTER
YOUR CHOICE OF SOLUTIONS.(:0:)')
                            END;
            ELSE      : BEGIN
                              SUCCESSFUL_INPUT:=FALSE;
                              WRITE_STRING('ERROR
IN YOUR INPUT. (:0:)')
                            END
          END "CASE OF MESSAGE IN"
        END; "WHILE"
       FOR I := 1 TO 25 DO  DISPLAY(NL);;
       IF((SOLUTION_SELECTED   =   1   )    OR
(SOLUTION_SELECTED = 3 ))
        THEN
          BEGIN "IF GRAPHICAL SOLUTION IS  TO
BE USED"
            WRITE_STRING('WHAT SCALE  DO  YOU
WISH ON YOUR GRAPH  (:0:)');
            WRITE_STRING('FOR THE  MEAN  TIME
TO FAILURE AXIS?  (:0:)');
            WRITE_STRING('ENTER D FOR DAYS, W
FOR WEEKS, OR M  (:0:)');
            WRITE_STRING('FOR MONTHS.   ENTER
YOUR CHOICE NOW.  (:0:)');
            FOR I := 1 TO 18 DO  DISPLAY(NL);
            SUCCESSFUL_INPUT:=FALSE;
            WHILE(NOT SUCCESSFUL_INPUT)DO
              BEGIN "WHILE"
                ACCEPT(MESSAGE_IN);
ACCEPT(DUMMY);
                CASE MESSAGE_IN OF
                     DAYS      : BEGIN
                                   SUCCESSFUL_INPUT:=TRUE;
                                   SCALE_SELECTED:=1
                                 END;
                     WEEKS     : BEGIN
                                   SUCCESSFUL_INPUT:=TRUE;
                                   SCALE_SELECTED:=2
                                 END;
                     MONTHS    : BEGIN
                                   SUCCESSFUL_INPUT:=TRUE;
                                   SCALE_SELECTED:=3
                                 END;
                     NEED_HELP : BEGIN
                                   SUCCESSFUL_INPUT:=FALSE;
                                   HELP_FLAG:=5;
                                   HELP(HELP_FLAG);
                                   WRITE_STRING('ENTER
```

```
GRAPHICAL SOLUTION SCALE DESIRED.(:0:)')
                                  END;
                        ELSE      : BEGIN
                                   SUCCESSFUL_INPUT:=FALSE;
                                   WRITE_STRING('ERROR
IN YOUR INPUT.(:0:)')
                                  END
                       END "CASE OF MESSAGE IN"
                      END; "WHILE"
                    FOR I := 1 TO 25 DO
DISPLAY(NL);
                  END; "IF GRAPHICAL SOLUTION IS
TO BE USED"
              "ENTER DATA FOR THE MODEL, OR
MODELS, SELECTED"
              SUCCESSFUL_INPUT:=FALSE;
              WHILE(NOT SUCCESSFUL_INPUT)DO
                BEGIN "WHILE"
                  WRITE_STRING('ENTER THE NUMBER
OF INITIAL ERRORS. (:0:)');
                  WRITE_STRING('IF YOU HAVE
SELECTED BOTH MODELS, YOU (:0:)');
                  WRITE_STRING('ARE REQUIRED TO
ONLY ENTER THE DATA ONCE. (:0:)');
                  FOR I := 1 TO 18 DO
DISPLAY(NL);
                  READ_INTEGER(NUMBER_IN);
                  IF(NUMBER_IN <= 0)
                   THEN
                     ERROR_COUNTER:=SUCC(ERROR_COUNTER)
                   ELSE
                     INITIAL_ERRORS:=NUMBER_IN;
                  WRITE_STRING('ENTER THE TOTAL
NUMBER OF INTERVALS. (:0:)');
                  FOR I := 1 TO 19 DO DISPLAY(NL);
                  READ_INTEGER(NUMBER_IN);
                  IF((NUMBER_IN <= 0) OR
(NUMBER_IN > MAX_ALLOWED) OR (NUMBER_IN >=
INITIAL_ERRORS))
                    THEN
                      ERROR_COUNTER:=SUCC(ERROR_COUNTER)
                    ELSE
                      BEGIN
                        TOTAL_INTERVALS:=NUMBER_IN;
                        ERRORS_DISCOVERED:=NUMBER_IN
                      END;
                  IF(ERROR_COUNTER <> 0)
                    THEN
                      BEGIN
                        ERROR_COUNTER:=0;
                        SUCCESSFUL_INPUT:=FALSE;
                        WRITE_STRING('ERROR IN
YOUR INPUT. (:0:)')
                      END
```

```
              ELSE
                SUCCESSFUL_INPUT := TRUE
            END; "WHILE"
          SUCCESSFUL_INPUT:=FALSE;
          WHILE(NOT SUCCESSFUL_INPUT)DO
            BEGIN "WHILE"
              WRITE_STRING('ENTER  THE  LENGTH
OF THE TIME INTERVALS  (:0:)');
              WRITE_STRING('SPECIFIED   ABOVE,
ONE AT A TIME. (:0:)');
              FOR   I   :=   1   TO   19   DO
DISPLAY(NL);
              FOR I:=1 TO TOTAL_INTERVALS DO
                BEGIN "FOR LOOP"
                  WRITE_STRING('ENTER       AN
INTERVAL NOW.  (:0:)');
                  READ_INTEGER(NUMBER_IN);
                  IF(NUMBER_IN <= 0)
                    THEN
                      ERROR_COUNTER:=SUCC(ERROR_COUNTER)
                    ELSE
                      BEGIN
                        IF((S_W_SELECTED)   OR
(ALL_SELECTED))
                          THEN
                            S_W_DATA_1(.I.):=NUMBER_IN;
                        IF((J_M_SELECTED)   OR
(ALL_SELECTED))
                          THEN
                            J_M_DATA_1(.I.):=NUMBER_IN
                      END
                END; "FOR LOOP"
              IF(ERROR_COUNTER <> 0)
                THEN
                  BEGIN
                    ERROR_COUNTER:=0;
                    SUCCESSFUL_INPUT:=FALSE;
                    WRITE_STRING('ERROR     IN
YOUR INPUT. (:0:)')
                  END
                ELSE
                  SUCCESSFUL_INPUT := TRUE
            END; "WHILE"
          IF((S_W_SELECTED) OR (ALL_SELECTED))
            THEN
              COMPUTE_SCHICK_WOLVERTON;
          IF((J_M_SELECTED) OR (ALL_SELECTED))
            THEN
              COMPUTE_JELINSKI_MORANDA;
          IF((SOLUTION_SELECTED   =   2)    OR
(SOLUTION_SELECTED = 3))
            THEN
              BEGIN
                IF((S_W_SELECTED)          OR
```

```
(ALL_SELECTED))
                  THEN
                    LOAD_TABULAR_DISPLAY_S_W;
                IF((J_M_SELECTED)           OR
(ALL_SELECTED))
                  THEN
                    LOAD_TABULAR_DISPLAY_J_M;
                PRINT_TABULAR_DISPLAY(S_W_SELECTED,J_M_SELEC
              END;
        IF((SOLUTION_SELECTED    =    1)    OR
(SOLUTION_SELECTED = 3))
          THEN
            BEGIN
              COLLECT_GRAPH_INFORMATION(S_W_SELECTED,J_M_SEL
              IF((S_W_SELECTED)             OR
(ALL_SELECTED))
                THEN
                  LOAD_S_W_GRAPH_DATA(SCALE_SELECTED);
              IF((J_M_SELECTED)             OR
(ALL_SELECTED))
                THEN
                  LOAD_J_M_GRAPH_DATA(SCALE_SELECTED);
              DRAW_GRAPH
            END
     END; "PROCEDURE PARTIAL PROMPTING"
```

```
   PROCEDURE PROVIDE_FULL_PROMPTING;

      "**********************************************************
      "*
*"
      "* THIS PROCEDURE  PERFORMS  EXACTLY THE
SAME FUNCTIONS AS   *"
      "* PROCEDURE PARTIAL PROMPTING, BUT WITH
MUCH MORE DETAIL  *"
      "* BEING SUPPLIED TO THE USER SO THAT HE
MAY SUCCESSFULLY  *"
      "* EXECUTE THE PROGRAM.  IT IS  DESIGNED
FOR THE LESS EX-   *"
      "*  PERIENCED  USER  OF  THIS   PROGRAM.
*"
      "*
*"
      "**********************************************************

     CONST
       BLANK                  = ' ';
       SCHICK_WOLVERTON       = 'S';
       JELINSKI_MORANDA       = 'J';
       BOTH                   = 'B';
       NEED_HELP              = 'H';
       DO_NOT_NEED_HELP       = 'N';
       GRAPHICAL              = 'G';
       TABULAR                = 'T';
       DAYS                   = 'D';
       WEEKS                  = 'W';
       MONTHS                 = 'M';

     VAR
       SUCCESSFUL_INPUT       : BOOLEAN;
       S_W_SELECTED           : BOOLEAN;
       J_M_SELECTED           : BOOLEAN;
       ALL_SELECTED           : BOOLEAN;
       HELP_FLAG              : INTEGER;
       NUMBER_IN              : INTEGER;
       ERROR_COUNTER          : INTEGER;
       SOLUTION_SELECTED      : INTEGER;
       SCALE_SELECTED         : INTEGER;
       I                      : INTEGER;
       MESSAGE_IN             : CHAR;
       DUMMY                  : CHAR;

      BEGIN "PROCEDURE PROVIDE FULL PROMPTING"
       SUCCESSFUL_INPUT      := FALSE;
       S_W_SELECTED          := FALSE;
       J_M_SELECTED          := FALSE;
```

```
        ALL_SELECTED          := FALSE;
        HELP_FLAG             := 0;
        INITIAL_ERRORS        := 0;
        TOTAL_INTERVALS       := 0;
        ERRORS_DISCOVERED     := 0;
        SOLUTION_SELECTED     := 0;
        SCALE_SELECTED        := 0;
        NUMBER_IN             := 0;
        ERROR_COUNTER         := 0;
        I                     := 0;
        MESSAGE_IN            := BLANK;
        DUMMY                 := BLANK;
        FOR I := 1 TO MAX_ALLOWED DO
          BEGIN
            S_W_DATA_1(.I.)   := 0;
            J_M_DATA_1(.I.)   := 0
          END;
           WRITE_STRING('YOU WILL  BE  PROVIDED
WITH A COMPLETE EXPLANATION(:0:)');
           WRITE_STRING('OF    QUESTIONS    AND
RESPONSES IN THE THIS PROMPTING MODE.(:0:)');
           WRITE_STRING('WHICH MODEL OR  MODELS
DO YOU DESIRE TO USE IN YOUR(:0:)');
           WRITE_STRING('SOLUTION.           THE
CANDIDATES ARE THE SCHICK-WOLVERTON,(:0:)');
           WRITE_STRING('THE  JELINSKI-MORANDA,
OR BOTH THESE MODELS.  EACH WILL(:0:)');
           WRITE_STRING('PROVIDE YOU  WITH  THE
MEAN TIME TO FAILURE AND THE RE-(:0:)');
           WRITE_STRING('LIABILITY   ASSOCIATED
WITH THE SOFTWARE PACKAGE WHICH(:0:)');
           WRITE_STRING('YOU  ENTER  DATA  FOR.
TO SELECT THE MODEL YOU DESIRE(:0:)');
           WRITE_STRING('TO USE, ENTER A SINGLE
LETTER AS FOLLOWS.  TO SELECT(:0:)');
           WRITE_STRING('THE   SCHICK-WOLVERTON
MODEL, ENTER AN S.  TO SELECT THE(:0:)');
           WRITE_STRING('JELINSKI-MORANDA
MODEL, ENTER A J.  TO SELECT BOTH OF(:0:)');
           WRITE_STRING('THE MODELS, ENTER A B.
TO INVOKE THE HELP PROCEDURE(:0:)');
           WRITE_STRING('ENTER AN H.(:0:)');
           FOR I := 1 TO 8 DO  DISPLAY(NL);
           SUCCESSFUL_INPUT:=FALSE;
           WHILE(NOT SUCCESSFUL_INPUT)DO
             BEGIN "WHILE"
               WRITE_STRING('WHICH  MODELS  DO
YOU DESIRE  TO  USE?  ENTER  AN  S,J,B, OR H.
(:0:)');
               ACCEPT(MESSAGE_IN);
ACCEPT(DUMMY);
               CASE MESSAGE_IN OF
                 SCHICK_WOLVERTON : BEGIN
                                      SUCCESSFUL_INPUT:=TRUE;
```

```
                                      S_W_SELECTED:=TRUE
                                    END;
               JELINSKI_MORANDA : BEGIN
                                      SUCCESSFUL_INPUT:=TRUE;
                                      J_M_SELECTED:=TRUE
                                    END;
               BOTH             : BEGIN
                                      SUCCESSFUL_INPUT:=TRUE;
                                      ALL_SELECTED:=TRUE
                                    END;
               NEED_HELP        : BEGIN
                                      SUCCESSFUL_INPUT:=FALSE
                                      HELP_FLAG:=3;
                                      HELP(HELP_FLAG)
                                    END;
               ELSE             : BEGIN
                                      SUCCESSFUL_INPUT:=FALSE
                                      WRITE_STRING('ERROR
IN YOUR INPUT. (:0:)')
                                    END
             END "CASE OF MESSAGE IN"
           END; "WHILE"
         FOR I := 1 TO 23 DO  DISPLAY(NL);
         WRITE_STRING('WHICH TYPE SOLUTION DO
YOU DESIRE AS YOUR OUT- (:0:)');
         WRITE_STRING('PUT.   THE  CANDIDATES
ARE A GRAPHICAL SOLUTION, (:0:)');
         WRITE_STRING('A TABULAR SOLUTION, OR
BOTH.  THE GRAPHICAL (:0:)');
         WRITE_STRING('SOLUTION PROVIDES YOUR
DATA PLOTTED AGAINST (:0:)');
         WRITE_STRING('THE  RELIABILITY   AND
THE MEAN TIME TO FAILURE. (:0:)');
         WRITE_STRING('THE   GRAPH   MAY   BE
PRESENTED ON ANY ONE OF THE (:0:)');
         WRITE_STRING('FOLLOWING TWO DEVICES,
THE CHROMATICS COLOR CRT, (:0:)');
         WRITE_STRING('OR     THE     PLOTTER
GRAPHICAL DISPLAY DEVICE.  (:0:)');
         WRITE_STRING('YOU  WILL   BE   ASKED
LATER WHICH DEVICE  (:0:)');
         WRITE_STRING('YOU  DESIRE   TO   USE
DURING THIS EXECUTION. (:0:)');
         WRITE_STRING('THE  TABULAR  SOLUTION
LISTS IN TABLE FORM THE (:0:)');
         WRITE_STRING('SOLUTIONS    OF    THE
PROGRAM.  TO SELECT ONLY THE (:0:)');
         WRITE_STRING('GRAPHICAL    SOLUTION,
ENTER G.  TO SELECT ONLY (:0:)');
         WRITE_STRING('THE TABULAR  SOLUTION,
ENTER T.  TO SELECT BOTH (:0:)');
         WRITE_STRING('TYPES  OF   SOLUTIONS,
ENTER B.  TO INVOKE THE (:0:)');
         WRITE_STRING('HELP PROCEDURE,  ENTER
```

```
H. (:0:)');
          FOR I := 1 TO 5 DO  DISPLAY(NL);
          SUCCESSFUL_INPUT:=FALSE;
          WHILE(NOT SUCCESSFUL_INPUT)DO
            BEGIN
              WRITE_STRING('WHICH   TYPE    OF
SOLUTION DO YOU  DESIRE?   ENTER  G,T,B, OR H.
(:0:)');
              ACCEPT(MESSAGE_IN);
ACCEPT(DUMMY);
              CASE MESSAGE_IN OF
                GRAPHICAL : BEGIN
                              SUCCESSFUL_INPUT:=TRUE;
                              SOLUTION_SELECTED:=1
                            END;
                TABULAR   : BEGIN
                              SUCCESSFUL_INPUT:=TRUE;
                              SOLUTION_SELECTED:=2
                            END;
                BOTH      : BEGIN
                              SUCCESSFUL_INPUT:=TRUE;
                              SOLUTION_SELECTED:=3
                            END;
                NEED_HELP : BEGIN
                              SUCCESSFUL_INPUT:=FALSE;
                              HELP_FLAG:=4;
                              HELP(HELP_FLAG)
                            END;
                ELSE      : BEGIN
                              SUCCESSFUL_INPUT:=FALSE;
                              WRITE_STRING('ERROR
IN YOUR INPUT. (:0:)')
                            END
              END "CASE OF MESSAGE IN"
            END; "WHILE"
          FOR I := 1 TO 23 DO  DISPLAY(NL);
          IF((SOLUTION_SELECTED   =   1)    OR
(SOLUTION_SELECTED = 3 ))
            THEN
              BEGIN "IF GRAPHICAL SOLUTION  IS
TO BE USED"
                WRITE_STRING('WHAT  SCALE   DO
YOU WISH ON YOUR GRAPH FOR THE(:0:)');
                WRITE_STRING('MEAN   TIME   TO
FAILURE AXIS?  THE POSSIBLE VALUES(:0:)');
                WRITE_STRING('ARE DAYS, WEEKS,
OR MONTHS.  THIS SCALE SHOULD BE(:0:)');
                WRITE_STRING('COMPATIBLE  WITH
YOUR DATA, THAT IS, IF THE DATA(:0:)');
                WRITE_STRING('YOU HAVE FOR THE
INTERVAL BETWEEN THE DISCOVERY(:0:)');
                WRITE_STRING('OF ERRORS IS  IN
DAYS, THEN YOUR SCALE SHOULD ALSO(:0:)');
                WRITE_STRING('BE IN DAYS.   IF
```

```
IT IS NOT, YOU MAY BE PRESENTED(:0:)');
                WRITE_STRING('WITH  A  WILDLY
SKEWED GRAPH.  TO SELECT DAYS, ENTER(:0:)');
                WRITE_STRING('D,   TO   SELECT
WEEKS, W, AND TO SELECT MONTHS,(:0:)');
                WRITE_STRING('ENTER AN M.   TO
INVOKE THE HELP PROCEDURE, ENTER(:0:)');
                WRITE_STRING('AN H.(:0:)');
                FOR  I   :=   1   TO   10   DO
DISPLAY(NL);
                SUCCESSFUL_INPUT:=FALSE;
                WHILE(NOT SUCCESSFUL_INPUT)DO
                  BEGIN "WHILE"
                    WRITE_STRING('WHAT   SCALE
DO YOU WISH  TO  USE?  ENTER  AN M,W,D, OR H.
(:0:)');
                    ACCEPT(MESSAGE_IN);
ACCEPT(DUMMY);
                    CASE MESSAGE_IN OF
                         DAYS      : BEGIN
                                       SUCCESSFUL_INPUT:=TRU
                                       SCALE_SELECTED:=1
                                     END;
                         WEEKS     : BEGIN
                                       SUCCESSFUL_INPUT:=TRU
                                       SCALE_SELECTED:=2
                                     END;
                         MONTHS    : BEGIN
                                       SUCCESSFUL_INPUT:=TRU
                                       SCALE_SELECTED:=3
                                     END;
                         NEED_HELP : BEGIN
                                       SUCCESSFUL_INPUT:=FAL
                                       HELP_FLAG:=5;
                                       HELP(HELP_FLAG)
                                     END;
                         ELSE      : BEGIN
                                       SUCCESSFUL_INPUT:=FAL
                                       WRITE_STRING('ERROR
IN YOUR INPUT. (:0:)')
                                     END
                       END "CASE OF MESSAGE
IN"
                     END; "WHILE"
                   FOR I  :=   1   TO  23  DO
DISPLAY(NL)
                 END; "IF GRAPHICAL  SOLUTION
IS TO BE USED"
          "ENTER  DATA  FOR   THE   MODEL,  OR
MODELS, SELECTED"
          WRITE_STRING('FOR  THE   MODEL   YOU
SELECTED, YOU MUST ENTER (:0:)');
          WRITE_STRING('THREE     TYPES     OF
INFORMATION.  YOU MUST ENTER (:0:)');
```

```
          WRITE_STRING('THE NUMBER OF  INITIAL
ERRORS ESTIMATED TO BE (:0:)');
          WRITE_STRING('PRESENT    IN     YOUR
SOFTWARE PACKAGE.  YOU HAVE (:0:)');
          WRITE_STRING('TO  ALSO   ENTER   THE
TOTAL NUMBER OF INTERVALS  (:0:)');
          WRITE_STRING('BETWEEN THE  DISCOVERY
OF AN ERROR.  FINALLY,  (:0:)');
          WRITE_STRING('MUST ENTER THE  LENGTH
OF EACH ERROR INTERVAL (:0:)');
          WRITE_STRING('OF   THE   TOTAL   YOU
SPECIFIED EARLIER.  YOU (:0:)');
          WRITE_STRING('ARE  RESTRICTED  TO  A
MAXIMUM NUMBER OF 250   (:0:)');
          WRITE_STRING('INTERVALS.  YOU DO NOT
ENTER THE TOTAL NUMBER  (:0:)');
          WRITE_STRING('OF  ERRORS  DISCOVERED
AS THE MODELS WILL ASSUME (:0:)');
          WRITE_STRING('ONLY  ONE  ERROR   WAS
DISCOVERED PER INTERVAL. (:0:)');
          WRITE_STRING('THE  TOTAL  NUMBER  OF
ERRORS DISCOVERED IS EQUAL (:0:)');
          WRITE_STRING('TO THE TOTAL NUMBER OF
INTERVALS SPECIFIED. (:0:)');
          WRITE_STRING('INSURE THAT YOUR  DATA
IS A POSITIVE INTEGER.  (:0:)');
          WRITE_STRING('IF ANY  PIECE  OF DATA
IS FOUND TO BE IN ERROR,  (:0:)');
          WRITE_STRING('YOU WILL  BE  REQUIRED
TO RE-ENTER THAT DATA. (:0:)');
          WRITE_STRING('INSURE THAT YOUR TOTAL
INTERVALS DOES NOT (:0:)');
          WRITE_STRING('EQUAL  OR  EXCEED  THE
INITIAL ERRORS ESTIMATED. (:0:)');
          WRITE_STRING('PRESS THE  RETURN  KEY
WHEN YOU ARE FINISHED (:0:)');
          WRITE_STRING('READING   THIS   PAGE.
(:0:)');
          DISPLAY(NL);
          ACCEPT(DUMMY);
          FOR I := 1 TO 23 DO  DISPLAY(NL);
          SUCCESSFUL_INPUT:=FALSE;
          WHILE(NOT SUCCESSFUL_INPUT)DO
            BEGIN  "WHILE"
              WRITE_STRING('DO YOU NEED  HELP?
IF YOU DO, ENTER H. (:0:)');
              WRITE_STRING('IF  YOU  DO   NOT,
ENTER N. (:0:)');
              FOR   I   :=   1   TO   20    DO
DISPLAY(NL);
              ACCEPT(MESSAGE_IN);
ACCEPT(DUMMY);
              CASE MESSAGE_IN OF
                NEED_HELP        : BEGIN
```

```
                                     SUCCESSFUL_INPUT:=TRUE;
                                     HELP_FLAG:=6;
                                     HELP(HELP_FLAG)
                                   END;
                DO_NOT_NEED_HELP                :
SUCCESSFUL_INPUT:=TRUE;
                ELSE               : BEGIN
                                     SUCCESSFUL_INPUT:=FALSE
                                     WRITE_STRING('ERROR
IN INPUT. (:0:)')
                                   END
              END "CASE OF MESSAGE_IN"
            END; "WHILE"
          FOR I := 1 TO 23 DO  DISPLAY(NL);
          SUCCESSFUL_INPUT:=FALSE;
          WHILE(NOT SUCCESSFUL_INPUT)DO
            BEGIN "WHILE"
              WRITE_STRING('ENTER  THE  NUMBER
OF INITIAL ERRORS. (:0:)');
              WRITE_STRING('IF    YOU    HAVE
SELECTED BOTH MODELS, YOU (:0:)');
              WRITE_STRING('ARE  REQUIRED   TO
ONLY ENTER THE DATA ONCE. (:0:)');
              FOR   I   :=   1   TO   18   DO
DISPLAY(NL);
              READ_INTEGER(NUMBER_IN);
              IF(NUMBER_IN <= 0)
               THEN
                 ERROR_COUNTER:=SUCC(ERROR_COUNTER)
               ELSE
                 INITIAL_ERRORS:=NUMBER_IN;
              WRITE_STRING('ENTER  THE   TOTAL
NUMBER OF INTERVALS. (:0:)');
              FOR   I   :=   1   TO   21   DO
DISPLAY(NL);
              READ_INTEGER(NUMBER_IN);
              IF((NUMBER_IN    <=    0)    OR
(NUMBER_IN > MAX_ALLOWED) OR (NUMBER_IN >=
INITIAL_ERRORS))
                THEN
                  ERROR_COUNTER:=SUCC(ERROR_COUNTER)
                ELSE
                  BEGIN
                    TOTAL_INTERVALS:=NUMBER_IN;
                    ERRORS_DISCOVERED:=NUMBER_IN
                  END;
              IF(ERROR_COUNTER <> 0)
                THEN
                  BEGIN
                    ERROR_COUNTER:=0;
                    SUCCESSFUL_INPUT:=FALSE;
                    WRITE_STRING('ERROR     IN
YOUR INPUT. (:0:)')
                  END
```

```
                ELSE
                  SUCCESSFUL_INPUT := TRUE
            END; "WHILE"
          SUCCESSFUL_INPUT:=FALSE;
          WHILE(NOT SUCCESSFUL_INPUT)DO
            BEGIN "WHILE"
              WRITE_STRING('ENTER  THE  LENGTH
OF THE TIME INTERVALS (:0:)');
              WRITE_STRING('YOU HAVE SPECIFIED
ABOVE, ONE AT A TIME. (:0:)');
              FOR  I  :=  1  TO  19  DO
DISPLAY(NL);
              FOR I:=1 TO TOTAL_INTERVALS DO
                BEGIN "FOR LOOP"
                  WRITE_STRING('ENTER       AN
INTERVAL NOW. (:0:)');
                  READ_INTEGER(NUMBER_IN);
                  IF(NUMBER_IN <= 0)
                    THEN
                      ERROR_COUNTER:=SUCC(ERROR_COUNTER)
                    ELSE
                      BEGIN
                        IF((S_W_SELECTED)   OR
(ALL_SELECTED))
                          THEN
                            S_W_DATA_1(.I.):=NUMBER_IN;
                        IF((J_M_SELECTED)   OR
(ALL_SELECTED))
                          THEN
                            J_M_DATA_1(.I.):=NUMBER_IN
                      END
                END; "FOR LOOP"
              IF(ERROR_COUNTER <> 0)
                THEN
                  BEGIN
                    ERROR_COUNTER:=0;
                    SUCCESSFUL_INPUT:=FALSE;
                    WRITE_STRING('ERROR     IN
YOUR INPUT. (:0:)')
                  END
                ELSE
                  SUCCESSFUL_INPUT := TRUE
            END; "WHILE"
          IF((S_W_SELECTED) OR (ALL_SELECTED))
            THEN
              COMPUTE_SCHICK_WOLVERTON;
          IF((J_M_SELECTED) OR (ALL_SELECTED))
            THEN
              COMPUTE_JELINSKI_MORANDA;
          IF((SOLUTION_SELECTED  =  2)   OR
(SOLUTION_SELECTED = 3))
            THEN
              BEGIN
                IF((S_W_SELECTED)          OR
```

```
(ALL_SELECTED))
                 THEN
                   LOAD_TABULAR_DISPLAY_S_W;
               IF((J_M_SELECTED)          OR
(ALL_SELECTED))
                 THEN
                   LOAD_TABULAR_DISPLAY_J_M;
               PRINT_TABULAR_DISPLAY(S_W_SELECTED,J_M_SELEC
             END;
         IF((SOLUTION_SELECTED   =   1)    OR
(SOLUTION_SELECTED = 3))
           THEN
             BEGIN
               COLLECT_GRAPH_INFORMATION(S_W_SELECTED,J_M_S
               IF((S_W_SELECTED)          OR
(ALL_SELECTED))
                 THEN
                   LOAD_S_W_GRAPH_DATA(SCALE_SELECTED);
               IF((J_M_SELECTED)          OR
(ALL_SELECTED))
                 THEN
                   LOAD_J_M_GRAPH_DATA(SCALE_SELECTED);
               DRAW_GRAPH
             END
      END; "PROCEDURE PROVIDE FULL PROMPTING"
```

```
        BEGIN "PROGRAM RELIABILITY MODEL"
            INITIAL_ERRORS                    :=
0;
            TOTAL_INTERVALS                   :=
0;
            ERRORS_DISCOVERED                 :=
0;
            I                                 :=
0;
            J                                 :=
0;
            HELP_FLAG                         :=
0;
            TAB_S_W_RELIABILITY               :=
0;
            TAB_S_W_MTTF                      :=
0;
            TAB_S_W_TIME_TO_FIND_ALL_ERRORS:=
0;
            TAB_S_W_STD                       :=
0;
            TAB_S_W_PHI                       :=
0;
            TAB_J_M_RELIABILITY               :=
0;
            TAB_J_M_MTTF                      :=
0;
            TAB_J_M_TIME_TO_FIND_ALL_ERRORS:=
0;
            TAB_J_M_STD                       :=
0;
            TAB_J_M_PHI                       :=
0;
            DESTINATION                       :=
0;
            SUCCESSFUL_INPUT                  :=
FALSE;
            J_M_TIME_TO_FIND_ALL_ERRORS       :=
0.0;
            S_W_TIME_TO_FIND_ALL_ERRORS       :=
0.0;
            J_M_STANDARD_DEVIATION            :=
0.0;
            S_W_STANDARD_DEVIATION            :=
0.0;
            S_W_PHI                           :=
0.0;
            J_M_PHI                           :=
0.0;
            ANSWER                            :=
0.0;
            S_W_RELIABILITY                   :=
```

```
0.0;
            S_W_MTTF                           :=
0.0;
            J_M_RELIABILITY                    :=
0.0;
            J_M_MTTF                           :=
0.0;
            MESSAGE_IN                         :=
BLANK;
            DUMMY                              :=
BLANK;
            FOR I := 1 TO MAX_ALLOWED DO
              BEGIN  "FOR LOOP"
                J_M_DATA_1(.I.)                :=
0;
                S_W_DATA_1(.I.)                :=
0
              END;   "FOR LOOP"
            FOR I := 1 TO MAX_ALLOWED DO
              BEGIN
                FOR J := 1 TO 2 DO
                  BEGIN
                    CHROM_DATA.S_W_PLOT(.I,J.)
:= 100;
                    CHROM_DATA.J_M_PLOT(.I,J.)
:= 100;
                    PLOT_DATA.S_W_PLOT(.I,J.)
:= 100;
                    PLOT_DATA.J_M_PLOT(.I,J.)
:= 100;
                    SPIN_DATA.S_W_PLOT(.I,J.)
:= 100;
                    SPIN_DATA.J_M_PLOT(.I,J.)
:= 100
                  END
              END;
            WRITE_STRING('WELCOME TO A PROGRAM
WHICH WILL COMPUTE THE (:0:)');
            WRITE_STRING('SOFTWARE RELIABILITY
OF A PARTIALLY DEBUGGED(:0:)');
            WRITE_STRING('SOFTWARE    PACKAGE.
THERE ARE TWO LEVELS OF  (:0:)');
            WRITE_STRING('INTERACTION
AVAILABLE TO YOU.  THESE ARE THE(:0:)');
            WRITE_STRING('FULL PROMPTING  MODE
AND THE PARTIAL PROMPTING(:0:)');
            WRITE_STRING('MODE.   IF  THIS  IS
YOUR FIRST EXECUTION OF THE(:0:)');
            WRITE_STRING('PROGRAM,  I  SUGGEST
YOU USE THE FULL OPTION.  (:0:)');
            WRITE_STRING('ADDITIONALLY,  THERE
IS A HELP PROCEDURE FOR  (:0:)');
            WRITE_STRING('YOUR          USE.
EXPLANATIONS FOR ITS USE ARE PRO- (:0:)');
```

```
            WRITE_STRING('VIDED   WITHIN   THE
PROMPTING OPTIONS.  ENTER AN(:0:)');
            WRITE_STRING('H AT  THIS  TIME  IF
YOU DESIRE HELP OR AN N IF (:0:)');
            WRITE_STRING('YOU DO NOT DESIRE  A
MORE DETAILED DESCRIPTION(:0:)');
            WRITE_STRING('OF   THIS   PROGRAM.
(:0:)');
            FOR I := 1 TO 9 DO  DISPLAY(NL);
            SUCCESSFUL_INPUT := FALSE;
            WHILE(NOT SUCCESSFUL_INPUT)DO
              BEGIN "WHILE"
                ACCEPT(MESSAGE_IN);
                ACCEPT(DUMMY);
                CASE MESSAGE_IN OF
                  NEED_HELP        : BEGIN
                                       SUCCESSFUL_INPUT:=TRU
                                       HELP_FLAG:=1;
                                       HELP(HELP_FLAG)
                                     END;
                  DO_NOT_NEED_HELP           :
SUCCESSFUL_INPUT:=TRUE;
                  ELSE             : BEGIN
                                       SUCCESSFUL_INPUT:=FAL
                                       WRITE_STRING('ERROR
IN YOUR INPUT.ENTER H OR N.(:0:)')
                                     END
                END "CASE MESSAGE IN"
              END; "WHILE"
            FOR I := 1 TO 23 DO  DISPLAY(NL);
            WRITE_STRING('WHICH    LEVEL    OF
PROMPTING DO YOU DESIRE?  ENTER(:0:)');
            WRITE_STRING('AN F  FOR  THE  FULL
PROMPTING OPTION OR A P FOR(:0:)');
            WRITE_STRING('THE          PARTIAL
PROMPTING OPTION.  YOU MAY ALSO(:0:)');
            WRITE_STRING('ENTER AN H FOR  HELP
IF YOU DESIRE.  PLEASE (:0:)');
            WRITE_STRING('ENTER YOUR CHOICE OF
OPTIONS NOW.(:0:)');
            FOR I := 1 TO 17 DO  DISPLAY(NL);
            SUCCESSFUL_INPUT := FALSE;
            WHILE(NOT SUCCESSFUL_INPUT)DO
              BEGIN "WHILE"
                ACCEPT(MESSAGE_IN);
                ACCEPT(DUMMY);
                CASE MESSAGE_IN OF
                  PARTIAL_LEVEL : BEGIN
                                    SUCCESSFUL_INPUT
:= TRUE;
                                    PARTIAL_PROMPTING
                                  END;
                  FULL_LEVEL    : BEGIN
                                    SUCCESSFUL_INPUT
```

```
:= TRUE;
                                                  PROVIDE_FULL_PROMPTING
                                               END;
                    NEED_HELP        : BEGIN
                                                  SUCCESSFUL_INPUT
:= FALSE;
                                                  HELP_FLAG
:= 2;
                                                  HELP(HELP_FLAG);
                                                  WRITE_STRING('PLEASE
SELECT YOUR PROMPTING OPTION  BY  ENTERING A P
OR AN F.(:0:)');
                                               END;
                    ELSE             : BEGIN
                                                  SUCCESSFUL_INPUT
:= FALSE;
                                                  WRITE_STRING('ERROR
IN  YOUR   INPUT.    SELECT   YOUR   PROMPTING
OPTION.(:0:)')
                                               END
                  END "CASE MESSAGE IN"
              END "WHILE"
          END.  "PROGRAM RELIABILITY MODEL"
```

```
"ROBERT  YOUNG"   "KANSAS   STATE  UNIVERSITY"
"DEPARTMENT OF COMPUTER SCIENCE"
  CONST COPYRIGHT  =  'COPYRIGHT  ROBERT YOUNG
1978' "########### # PREFIX  # ###########"  ;
CONST NL = '(:10:)'
  ; FF = '(:12:)'
  ; CR = '(:13:)'
  ; EM = '(:25:)' ;  CONST  PAGELENGTH = 512 ;
TYPE PAGE =  ARRAY  (.  1  .. PAGELENGTH .) OF
CHAR ; CONST  LINELENGTH  =  132 ; TYPE LINE =
ARRAY (. 1 .. LINELENGTH .) OF CHAR;


    "#########################################################
    "#
#"
    "# THE FOLLOWING RECORD TYPE IS THE  MEANS
BY WHICH ALL   #"
    "# DATA IS PASSED  TO  THIS PROGRAM.  THIS
PROGRAM HAS     #"
    "# BEEN TREATED  AS  AN EXTERNAL PROCEDURE
AND THIS FORM  #"
    "# OF  A  RECORD  IS  THE PARAMETER OF THE
PROGRAM CALL.    #"
    "#
#"
    "#  AN   INSTANCE   OF   THIS   RECORD  IS
INSTANTIATED IN THE    #"
    "# PROGRAM HEADING.  THE VARIABLE DECLARED
IS PARAM.    #"
    "#
#"
    "# ALL IDENTIFIERS USED WITHIN THE  RECORD
SHOULD BE      #"
    "# SELF-EXPLANATORY  AND  NEED  NO FURTHER
EXPLANATION.    #"
    "#
#"
    "#########################################################


  TYPE COORDINATE = RECORD
                      S_W_PLOT
: ARRAY(.1..250,1..2.) OF INTEGER;
                      J_M_PLOT
: ARRAY(.1..250,1..2.) OF INTEGER;
                      COMBINATION_SELECTED
: INTEGER;
                      SCALE_DESIRED
: INTEGER;
                      NUMBER_OF_ERRORS_OBSERVED
: INTEGER
                    END;
```

```
 CONST IDLENGTH = 12 ; TYPE IDENTIFIER = ARRAY
(. 1 .. IDLENGTH .) OF CHAR ; TYPE FILE = 1 ..
2 ; TYPE FILEKIND = ( EMPTY . SCRATCH ,  ASCII
, SEQCODE , CONCODE ) ; TYPE FILEATTR
    = RECORD
        KIND : FILEKIND
      ; ADDR : INTEGER
      ; PROTECTED : BOOLEAN
      ; NOTUSED  :  ARRAY  (.  1  ..  5  .) OF
INTEGER
      END ; TYPE IODEVICE
    = ( TYPEDEVICE , DISKDEVICE , TAPEDEVICE ,
PRINTDEVICE
      , CARDDEVICE )  ;  TYPE  IOOPERATION = (
INPUT , OUTPUT , MOVE , CONTROL ) ; TYPE IOARG
= ( WRITEEOF , REWIND , UPSPACE , BACKSPACE  )
; TYPE IORESULT
    = ( COMPLETE , INTERVENTION , TRANSMISSION
, FAILURE . ENDFILE
      ,  ENDMEDIUM  ,  STARTMEDIUM  )  ;  TYPE
IOPARAM
    = RECORD
        OPERATION : IOOPERATION
      ; STATUS : IORESULT
      ; ARG : IOARG
      END ;  TYPE  TASKKIND  =  (  INPUTTASK ,
JOBTASK , OUTPUTTASK  )  ;  TYPE  ARGTAG  = (
NILTYPE  ,  BOOLTYPE  ,  INTTYPE  ,  IDTYPE  ,
PTRTYPE ) ;  TYPE  POINTER  = @ BOOLEAN ; TYPE
PASSPTR = @ PASSLINK ; TYPE PASSLINK
    = RECORD
        OPTIONS : SET OF CHAR
      ; FILLER1  :  ARRAY  (.  1  ..  7  .) OF
INTEGER
      ; FILLER2 : BOOLEAN
      ; RESET_POINT : INTEGER
      ; FILLER3  :  ARRAY  (.  1  ..  11 .) OF
POINTER
      END ; TYPE ARGTYPE
    = RECORD
        CASE TAG : ARGTAG
        OF NILTYPE  ,  BOOLTYPE  :  (  BOOL  :
BOOLEAN )
          ; INTTYPE : ( INT : INTEGER )
          ; IDTYPE : ( ID : IDENTIFIER )
          ; PTRTYPE : ( PTR : PASSPTR )
      END ; CONST MAXARG = 10 ; TYPE ARGLIST =
ARRAY (.  1  ..  MAXARG  .)  OF ARGTYPE ; TYPE
ARGSEQ = ( INP . OUT ) ; TYPE PROGRESULT
    = ( TERMINATED . OVERFLOW , POINTERERROR ,
RANGEERROR
      , VARIANTERROR , HEAPLIMIT ,  STACKLIMIT
, CODELIMIT . TIMELIMIT
      , CALLERROR ) ; PROCEDURE READ ( VAR C :
```

```
CHAR ) ; PROCEDURE WRITE ( C : CHAR ) ;
PROCEDURE OPEN ( F : FILE ; ID : IDENTIFIER ;
VAR FOUND : BOOLEAN ) ; PROCEDURE CLOSE ( F :
FILE ) ; PROCEDURE GET ( F : FILE ; P :
INTEGER ; VAR BLOCK : UNIV PAGE ) ; PROCEDURE
PUT ( F : FILE ; P : INTEGER ; VAR BLOCK :
UNIV PAGE ) ; FUNCTION LENGTH ( F : FILE ) :
INTEGER ; PROCEDURE MARK ( VAR TOP : INTEGER )
; PROCEDURE RELEASE ( TOP : INTEGER ) ;
PROCEDURE IDENTIFY ( HEADER : LINE ) ;
PROCEDURE ACCEPT ( VAR C : CHAR ) ; PROCEDURE
DISPLAY ( C : CHAR ) ; PROCEDURE NOTUSED ;
PROCEDURE NOTUSED2 ; PROCEDURE NOTUSED3 ;
PROCEDURE NOTUSED4 ; PROCEDURE NOTUSED5 ;
PROCEDURE NOTUSED6 ; PROCEDURE NOTUSED7 ;
PROCEDURE NOTUSED8 ; PROCEDURE NOTUUED9 ;
PROCEDURE NOTUSED10 ; PROCEDURE RUN
    ( ID : IDENTIFIER
    ; VAR PARAM : ARGLIST
    ; VAR LINE : INTEGER
    ; VAR RESULT : PROGRESULT
    );


  PROGRAM CHROFIX ( PARAM : COORDINATE );


  "#############################################################
  "#
#"
  "# PROGRAM CHROFIX WAS ORIGINALLY AUTHORED
BY THEORDORE JOHN      #"
  "# SOCOFLOSKY, A FORMER GRADUATE STUDENT AT
KSU. WITH THE EXCEP-#"
  "#   TION   OF   PROCEDURES   USER_PROG,
SET_UP_SYSTEM, DRAW_GRAPH, AND  #"
  "# OUTPUT_LINE, ALL PROCEDURES USED IN THIS
PROGRAM WERE WRITTEN #"
  "# BY SOCOFLOSKY.  THE PURPOSE OF THESE
PROCEDURES IS TO PROVIDE #"
  "# THE NECESSARY INTERFACE BETWEEN THE
INTERDATA COMPUTER AND THE#"
  "# CHROMATICS COLOR DISPLAY DEVICE AND TO
PROVIDE ANY PROGRAMMER #"
  "# WITH THE CAPABILITY TO OPERATE THE DEVICE
REMOTELY.          #"
  "#
#"
  "# THE PROCEDURES MENTIONED ABOVE AS THE
EXCEPTIONS WERE AUTHORED#"
  "# BY MYSELF AND PROVIDE THE MECHANISMS TO
DRAW THE GRAPHICAL     #"
  "# SOLUTION ITSELF ON THE CHROMATICS CRT
FACE.                  #"
```

```
  "*
*"
  "*************************************************************


"*************************************************************
"*
*" "* INTERDATA OPERATING SYSTEM 32MT3 SVC
INTERFACE ROUTINE TYPES *" "*
*"
"*************************************************************

"MISCELLANEOUS DATA TYPES"
    TYPE CHAR1 = PACKED ARRAY [ 1 .. 1 ] OF
CHAR
  ; TYPE CHAR3 = PACKED ARRAY [ 1 .. 3 ] OF
CHAR
  ; TYPE CHAR8 = PACKED ARRAY [ 1 .. 8 ] OF
CHAR
  ; TYPE CHAR4 = PACKED ARRAY [ 1 .. 4 ] OF
CHAR
  ; TYPE CHAR16 = ARRAY [ 1 .. 16 ] OF CHAR
  ; TYPE CHAR28 = ARRAY [ 1 .. 28 ] OF CHAR
"SVC1 PARAMETER BLOCK"
  ; TYPE SVC1_BLOCK
       = RECORD
           SVC1_FUNC : BYTE "FUNCTION CODE"
         ; SVC1_LU : BYTE "LOGICAL UNIT NUMBER"
         ; SVC1_STAT : BYTE "DEV-INDEP STATUS"
         ; SVC1_DEV_STAT : BYTE "DEV-DEPENDENT
STATUS"
         ; SVC1_BUFSTART : INTEGER
"ADDRESS(BUFFER)"
         ; SVC1_BUFEND : INTEGER
"ADDRESS(BUFFER)+SIZE(BUFFER)-1"
         ; SVC1_RANDOM_ADDR : INTEGER "RANDOM
ADDRESS FOR DASD"
         ; SVC1_XFER_LEN : INTEGER "TRANSFER
LENGTH"
         ; SVC1_RESERVED : INTEGER "RESERVED
FOR ITAM USE"
         ;
         END "SVC 1 FUNCTION CODES"
  ; CONST SVC1_DATA_XFER = #00
    ; SVC1_COMMAND = #80
    ; SVC1_READ = #40
    ; SVC1_WRITE = #20
    ; SVC1_TESTSET = #60
    ; SVC1_TESTIO = #00
    ; SVC1_ASCII = #00
    ; SVC1_BINARY = #10
    ; SVC1_PROCEED = #00
    ; SVC1_WAIT = #08
    ; SVC1_SEQL = #00
```

```
    ; SVC1_RANDOM = #04
    ; SVC1_CWAIT = #00
    ; SVC1_UNC_PROC = #02
    ; SVC1_FORMAT = #00
    ; SVC1_IMAGE = #01
    ; SVC1_REW = #40
    ; SVC1_BSR = #20
    ; SVC1_FSR = #10
    ; SVC1_WFM = #08
    ; SVC1_FSF = #04
    ; SVC1_BSF = #02
    ;    SVC1_RESV_FN    =    #01    "SVC    1
DEVICE-INDEPENDENT STATUS CODES"
  ; CONST SVC1_OK = #00
    ; SVC1_ERROR = #80
    ; SVC1_ILGFN = #40
    ; SVC1_DU = #20
    ; SVC1_EOM = #10
    ; SVC1_EOF = #08
    ; SVC1_UNRV = #04
    ; SVC1_RECV = #02
    ; SVC1_ILGLU = #01
    ; SVC1_DEVBUSY = #7F "FILE DESCRIPTOR  FOR
SVC7 REQUESTS"
  ; TYPE FD_TYPE
      = PACKED RECORD
                VOLN : CHAR4 "VOLUME NAME"
              ; FN : CHAR8 "FILE NAME"
              ; EXTN : CHAR3 "EXTENSION"
              ; ACCT :  CHAR  "ACCOUNT NUMBER
CODE"
              ;
              END "SVC 7 PARAMETER BLOCK"
  ; TYPE SVC7_BLOCK
      = RECORD
          SVC7_CMD : BYTE "COMMAND"
        ;  SVC7_MOD  :  BYTE  "MODIFIER/DEVICE
TYPE"
        ; SVC7_STAT : BYTE "STATUS"
        ; SVC7_LU : BYTE "LOGICAL UNIT NUMBER"
        ; SVC7_KEYS : SHORTINTEGER "READ/WRITE
KEYS"
        ; SVC7_RECLEN : SHORTINTEGER  "LOGICAL
RECORD LENGTH"
        ; SVC7_FD : FD_TYPE "FILE DESCRIPTOR"
        ; SVC7_SIZE  :  INTEGER  "FILE(/INDEX)
SIZE"
        ;
        END "SVC 7 COMMAND CODES"
  ; CONST SVC7_ALLOC = #80
    ; SVC7_ASSIGN = #40
    ; SVC7_CHAP = #20
    ; SVC7_RENAME = #10
    ; SVC7_REPROT = #08
```

```
    ; SVC7_CLOSE = #04
    ; SVC7_DELETE = #02
    ; SVC7_CHECKPT = #01
    ; SVC7_FETCH_ATTR  =  #00  "SVC 7 MODIFIER
CODES - ACCESS PRIVILEGES"
  ; CONST SVC7_AP_SRO = #00
    ; SVC7_AP_ERO = #20
    ; SVC7_AP_SWO = #40
    ; SVC7_AP_EWO = #60
    ; SVC7_AP_SRW = #80
    ; SVC7_AP_SREW = #A0
    ; SVC7_AP_ERSW = #C0
    ; SVC7_AP_ERW = #E0 "SVC 7 MODIFIER  CODES
- BUFFERING/FILE TYPE"
  ; CONST SVC7_BUF_DEFAULT = #00
    ; SVC7_BUF_PHYS = #08
    ; SVC7_BUF_LOG = #10
    ; SVC7_BUF_SVC15 = #18
    ; SVC7_FTYPE CONTIG = #00
    ; SVC7_FTYPE CHAIN = #01
    ; SVC7_FTYPE INDEX = #02
    ; SVC7_FTYPE_ITAM = #07


  ; CONST BLACK = 0
    ; BLUE = 1
    ; GREEN = 2
    ; CYAN = 3
    ; RED = 4
    ; MAGENTA = 5
    ; YELLOW = 6
    ; WHITE = 7
    ; ORANGE = 8
    ; PURPLE = 9


  ; VAR ALINE : LINE
    ; J , TEMPX , TEMPY : INTEGER
    ; OUTLINE : ARRAY [ 1 .. 132 ] OF CHAR
    ; OUTPUT_COUNTER : INTEGER
    ; C : CHAR
    ; X , Y  ,  X1  ,  Y1 , X2 , Y2 , RADIUS :
INTEGER
    ; COLOR_NUM : INTEGER
    ; SVC1_OUT : SVC1_BLOCK;


"*****************     USEFULL      PROCEDURES
**********************"

   PROCEDURE SVC1 ( SV_OUT : SVC1_BLOCK );
     EXTERN;

   PROCEDURE OUTPUT(C : CHAR);
```

```
  FORWARD;

PROCEDURE OUTPUT_DEC(I : INTEGER);
  VAR I100 : INTEGER;
    BEGIN "PROCEDURE OUTPUT DECIMAL"
      I100 := I DIV 100;
      OUTPUT(CHR(I100 + 48));
      OUTPUT(CHR(((I - I100 * 100) DIV 10 )
+ 48 ));
      OUTPUT(CHR((I MOD 10) + 48))
    END; "PROCEDURE OUTPUT DECIMAL"

PROCEDURE INIT_SVC;
  BEGIN "INIT SVC CALLS"
     WITH SVC1_OUT DO
       BEGIN
         SVC1_FUNC    :=    SVC1_WRITE    +
SVC1_IMAGE;
         SVC1_LU := 0;
         SVC1_STAT := 0;
         SVC1_DEV_STAT := 0;
         SVC1_BUFSTART := ADDRESS ( OUTLINE
)
       END;
     OUTPUT_COUNTER := 0
   END; "PROCEDURE INIT SVC CALLS"

PROCEDURE MODE;
  BEGIN "PROCEDURE MODE"
    OUTPUT(CHR(1))
  END; "PROCEDURE MODE"

 PROCEDURE OUTPUT_COORD(X,Y : INTEGER);
   BEGIN "PROCEDURE OUTPUT COORDINATE"
     OUTPUT_DEC(X);
     OUTPUT_DEC(Y)
   END; "PROCEDURE OUTPUT COORDINATE"

PROCEDURE SHIP_OUT;
  BEGIN "PROCEDURE SHIP OUT"
     SVC1_OUT . SVC1_BUFEND
     :=   SVC1_OUT   .   SVC1_BUFSTART    +
OUTPUT_COUNTER - 1 ;
     SVC1 ( SVC1_OUT );
    OUTPUT_COUNTER := 0
   END; "PROCEDURE SHIP OUT"

PROCEDURE OUTPUT ( C : CHAR );
  BEGIN "PROCEDURE OUTPUT"
     OUTPUT_COUNTER     :=      SUCC      (
OUTPUT_COUNTER );
     OUTLINE [ OUTPUT COUNTER ] := C
   END; "PROCEDURE OUTPUT"
```

```
   PROCEDURE MOVE_CURSOR ( X , Y : INTEGER  );
"3.6.3.5"
     BEGIN "PROCEDURE MOVE CURSOR"
        MODE;
        OUTPUT ( 'U' );
        OUTPUT_COORD ( X , Y );
        SHIP_OUT
      END; "PROCEDURE MOVE CURSOR"

   PROCEDURE  SELECT COLOR   (   COLOR_NUM   :
INTEGER ); "3.7.4.1"
     BEGIN" PROCEDURE SELECT COLOR"
        MODE;
        OUTPUT ( 'C' );
        OUTPUT ( CHR ( COLOR_NUM + 48 ) );
        SHIP_OUT
      END; "PROCEDURE SELECT COLOR"

   PROCEDURE BACKGROUND_ON; "3.7.4.3"
      BEGIN "PROCEDURE BACKGROUND ON"
        MODE;
        OUTPUT('M');
        SHIP_OUT
      END; "PROCEDURE BACKGROUND ON"

   PROCEDURE BACKGROUND_OFF; "3.7.4.4"
     BEGIN "PROCEDURE BACKGROUND OFF"
       MODE;
       OUTPUT('N');
       SHIP_OUT
     END; "PROCEDURE BACKGROUND OFF"

   PROCEDURE ERASE_PAGE; "3.7.5.1"
     BEGIN "PROCEDURE ERASE PAGE"
       OUTPUT(CHR(12));
       SHIP_OUT
     END; "PROCEDURE ERASE PAGE"

   PROCEDURE PLOT_ON; "3.9.1"
     BEGIN "PROCEDURE PLOT ON"
       MODE;
       OUTPUT('G');
       SHIP_OUT
     END; "PROCEDURE PLOT ON"

   PROCEDURE PLOT_OFF; "3.9.2"
     BEGIN "PROCEDURE PLOT OFF"
       OUTPUT(CHR(21));
       SHIP_OUT
     END; "PROCEDURE PLOT OFF"

   PROCEDURE VECTOR  (  X1  ,  Y1  , X2 , Y2 :
INTEGER ); "3.9.5.7"
     BEGIN "PROCEDURE VECTOR"
```

```
    OUTPUT(CHR(39));
    OUTPUT_COORD(X1,Y1);
    OUTPUT_COORD(X2,Y2);
    SHIP_OUT
  END; "PROCEDURE VECTOR"


"*************************************************************
 *
*
 * THE FOLLOWING PROCEDURES AREN'T  PRIMITIVES
OF THE CHROMATICS *
 * THEY ARE ADDED  FOR  THE CONVENIENCE OF THE
PROGRAMMER.        *
 *
*
 ************************************************************

   PROCEDURE FIX ( K : INTEGER );
     BEGIN "PROCEDURE FIX"
       FOR J:= K DOWNTO 0 DO
         "NOTHING BUT DELAY THE INTERDATA"
     END; "PROCEDURE FIX"

    FUNCTION DCOS(RADIANS : REAL) : REAL;
      FORTRAN;

   FUNCTION DSIN ( RADIANS : REAL ) : REAL;
     FORTRAN;
```

```
PROCEDURE DRAW_GRAPH(WHICH_ONE : INTEGER);

    "#########################################################
    "#
#"
    "# THIS PROCEDURE IS USED TO DRAW THE
GRAPHS ON THE     #"
    "# CHROMATICS CRT. THE PROCEDURE
ESSENTIALLY CONTROLS #"
    "# PRIMITIVE OPERATIONS AND INCLUDES A
SIMPLE ROUTINE   #"
    "# TO OBTAIN THE COORDINATES FROM THE
PASSED RECORD.    #"
    "#
#"
    "# X-AXIS COORDINATES ARE CONTAINED IN
COLUMN ONE OF     #"
    "# OF THE PLOT ARRAYS AND THE Y-AXIS
COORDINATES ARE    #"
    "# CONTAINED IN THE SECOND COLUMN.
#"
    "# DRAWING THE GRAPH THEN BECOMES ONLY A
SIMPLE MATTER  #"
    "# OF VECTORING FROM ONE SET OF
COORDINATES TO THE NEXT.#"
    "#
#"
    "# THE CASE STATEMENT INDICATES THROUGH
THE VALUES OF    #"
    "# THE VARIABLE WHICH_ONE WHICH MODEL'S
GRAPH TO DRAW.  #"
    "# A VALUE OF 100 REFERENCES THE
SCHICK-WOLVERTON MODEL,#"
    "# 200 REFERENCES THE JELINSKI-MORANDA
MODEL, AND 300   #"
    "# AND 400 REFERENCE BOTH MODELS BEING
SELECTED.        #"
    "#
#"
    "#########################################################


  VAR I : INTEGER;
    BEGIN "DRAW GRAPH"
      PLOT_OFF;
      FIX(300000);
      CASE WHICH_ONE OF
        100,300 :  BEGIN
                     SELECT_COLOR(RED);
                     WITH PARAM DO
                       BEGIN
                         MOVE_CURSOR(100,100);
                         PLOT_ON;
```

```
                              I := 1;
                              VECTOR(100,100,S_W_PLOT(.I,1.),S_
                              IF(NUMBER_OF_ERRORS_OBSERVED
> 1)
                                THEN
                                  FOR I  :=  2 TO
NUMBER_OF_ERRORS_OBSERVED DO
                                    VECTOR(S_W_PLOT(.I-1,1.),S_
                              PLOT_OFF;
                              MOVE_CURSOR(0,0)
                            END
                        END;
          200,400 :  BEGIN
                          SELECT_COLOR(WHITE);
                          WITH PARAM DO
                            BEGIN
                              MOVE_CURSOR(100,100);
                              PLOT_ON;
                              I := 1;
                              VECTOR(100,100,J_M_PLOT(.I,1.),J_
                              IF(NUMBER_OF_ERRORS_OBSERVED
> 1)
                                THEN
                                  FOR I  :=  2 TO
NUMBER_OF ERRORS_OBSERVED DO
                                    VECTOR(J_M_pLOT(.I-1,1.),J_
                              PLOT_OFF;
                              MOVE_CURSOR(0,0)
                            END
                         END
        END "CASE OF WHICH ONE"
      END; "DRAW GRAPH"
```

```
 PROCEDURE OUTPUT_LINE(ALINE : LINE);


      "*******************************************************
      "*
*"
      "* THIS PROCEDURE  IS  USED  TO MARK THE
SCALE OF THE X   *"
      "* AND Y AXIS OF THE GRAPH AND TO  WRITE
THE LEGENDS.   *"
      "*
*"
      "*******************************************************


      VAR I : INTEGER;
        BEGIN "PROCEDURE OUTPUT LINE"
          I := 1;
          WHILE(ALINE(.I.) <> '(:0:)') DO
            BEGIN
              OUTPUT(ALINE(.I.));
              I := SUCC(I)
            END;
          SHIP_OUT
        END; "PROCEDURE OUTPUT LINE"
```

```
PROCEDURE SET_UP_SYSTEM;

    "*****************************************************
    "*
*"
    "* THIS PROCEDURE IS USED TO  INITIALIZE
THE CHROMATICS       *"
    "* CRT  FACE.   THE  GRAPHICAL  FRAME IS
INITIALIZED HERE, AS   *"
    "*  WELL  AS   THE   LEGENDS  AND  SCALE
MARKINGS.  THE ORIGIN OF   *"
    "*    THE     GRAPH     IS     100,100.
*"
    "*
*"
    "*****************************************************


    BEGIN "SET UP SYSTEM"
      BACKGROUND_ON;
      SELECT COLOR(BLUE);
      BACKGROUND_OFF;
      FIX(300000);
      ERASE_PAGE;
      FIX(300000);
      PLOT_ON;
      SELECT_COLOR(RED);

      "SET UP X AXIS OF CHROMATICS CRT"

      VECTOR(100,100,400,100);
      VECTOR(100,100,100,465);
      VECTOR(130,100,130,094);
      VECTOR(160,100,160,094);
      VECTOR(190,100,190,094);
      VECTOR(220,100,220,094);
      VECTOR(250,100,250,094);
      VECTOR(280,100,280,094);
      VECTOR(310,100,310,094);
      VECTOR(340,100,340,094);
      VECTOR(370.100,370.094);
      VECTOR(400,100,400,094);

      "SET UP Y-AXIS ON CHROMATICS CRT"

      VECTOR(100,130,094,130);
      VECTOR(100,160,094,160);
      VECTOR(100,190.094,190);
      VECTOR(100,220,094,220);
      VECTOR(100.250,094,250);
      VECTOR(100,280,094,280);
      VECTOR(100,310,094,310);
      VECTOR(100,340,094,340);
```

```
VECTOR(100,370.094,370);
VECTOR(100,400,094,400);
VECTOR(100,450,094,450);
PLOT_OFF;

"LABEL THE X-AXIS FOR RELIABILITY"

FIX(300000);
MOVE_CURSOR(125.88);
OUTPUT_LINE('.1(:0:)');
MOVE_CURSOR(155,88);
OUTPUT_LINE('.2(:0:)');
MOVE_CURSOR(185,88);
OUTPUT_LINE('.3(:0:)');
MOVE_CURSOR(215,88);
OUTPUT_LINE('.4(:0:)');
MOVE_CURSOR(245,88);
OUTPUT_LINE('.5(:0:)');
MOVE_CURSOR(275,88);
OUTPUT_LINE('.6(:0:)');
MOVE_CURSOR(305,88);
OUTPUT_LINE('.7(:0:)');
MOVE_CURSOR(335,88);
OUTPUT_LINE('.8(:0:)');
MOVE_CURSOR(365,88);
OUTPUT_LINE('.9(:0:)');
MOVE_CURSOR(395,88);
OUTPUT_LINE('1.0(:0:)');

"LABEL THE Y-AXIS FOR MTTF"

MOVE_CURSOR(078,135);
OUTPUT_LINE('2(:0:)');
MOVE_CURSOR(078,165);
OUTPUT_LINE('4(:0:)');
MOVE_CURSOR(078,195);
OUTPUT_LINE('6(:0:)');
MOVE_CURSOR(078,225);
OUTPUT_LINE('8(:0:)');
MOVE_CURSOR(078,255);
OUTPUT_LINE('10(:0:)');
MOVE_CURSOR(078,285);
OUTPUT_LINE('12(:0:)');
MOVE_CURSOR(078,315);
OUTPUT_LINE('14(:0:)');
MOVE_CURSOR(078,345);
OUTPUT_LINE('16(:0:)');
MOVE_CURSOR(078,375);
OUTPUT_LINE('18(:0:)');
MOVE_CURSOR(078,405);
OUTPUT_LINE('20(:0:)');
MOVE_CURSOR(072,455);
OUTPUT_LINE('>20(:0:)');
```

```
        "LABEL THE Y-AXIS AS MTTF"

        MOVE_CURSOR(028,320);
        OUTPUT_LINE('MTTF(:0:)');
        MOVE_CURSOR(028,290);
        OUTPUT_LINE(' IN(:0:)');
        MOVE_CURSOR(028,260);
        IF(PARAM.SCALE_DESIRED = 1)
          THEN
            OUTPUT_LINE('DAYS(:0:)')
          ELSE
            IF(PARAM.SCALE_DESIRED = 2)
              THEN
                OUTPUT_LINE('WEEKS(:0:)')
              ELSE
                OUTPUT_LINE('MONTHS(:0:)');

        "LABEL THE X-AXIS AS RELIABILITY"

        MOVE_CURSOR(220,070);
        OUTPUT_LINE('RELIABILITY(:0:)');

        "PUT NOTE IN  REFERENCE  COLOR OF EACH
MODEL"
        "AND  MOVE  THE  CURSOR  BACK  TO  THE
ORIGIN."

        MOVE_CURSOR(160,050);
        OUTPUT_LINE('SCHICK-WOLVERTON DATA  IS
IN RED.(:0:)');
        SELECT COLOR(WHITE);
        MOVE_CURSOR(160,038);
        OUTPUT_LINE('JELINSKI-MORANDA DATA  IS
IN WHITE.(:0:)');
        MOVE_CURSOR(0,0)
      END; "PROCEDURE SET UP SYSTEM"
```

```
PROCEDURE USER_PROG;

    "#########################################################
    "#
#"
    "# THIS PROCEDURE DETERMINES WHICH  MODEL
HAS BEEN CHOSEN    #"
    "# TO PRESENT A  GRAPH  ON THE CHROMTICS.
THE KEY IS THE     #"
    "# VARIABLE  COMBINATION_SELECTED,  WHICH
IS PASSED BY THE   #"
    "#  CALLING   PROGRAM.   THIS   VARIABLE
COINCIDES WITH PROGRAM  #"
    "#         PROJECT'S          DESTINATION.
#"
    "#
#"
    "#########################################################

    VAR WHICH_ONE : INTEGER;
      BEGIN "PROCEDURE USER PROGRAM"
        SET_UP_SYSTEM;
        CASE PARAM.COMBINATION_SELECTED OF
          1,10.13,19 : BEGIN
                         WHICH_ONE := 100;
                         DRAW_GRAPH(WHICH_ONE)
                       END;
          2,11,14,20 : BEGIN
                         WHICH_ONE := 200;
                         DRAW_GRAPH(WHICH_ONE)
                       END;
          3.12.15.21 : BEGIN
                         WHICH_ONE := 300;
                         DRAW_GRAPH(WHICH_ONE);
                         WHICH_ONE := 400;
                         DRAW_GRAPH(WHICH_ONE)
                       END
        END; "CASE OF COMBINATION SELECTED"
        PLOT_OFF
      END;  "PROCEDURE USER PROGRAM"



BEGIN "PROGRAM CHROFIX"
  INIT_SVC;
  USER_PROG
END.  "PROGRAM CHROFIX"
```

```
"ROBERT   YOUNG"   "DEPARTMENT   OF   COMPUTER
SCIENCE"
  CONST COPYRIGHT  =  'COPYRIGHT  ROBERT YOUNG
1978' "/////////// #  PREFIX  # ###########" ;
CONST NUL = '(:00:)'
  ; MODE = '(:01:)'
  ; X_BAR = '!'
  ; BS = '(:08:)'
  ; NAK = '(:15:)'
  ; Y_BAR = '"'
  ; NL = '(:10:)'
  ; A = 'A'
  ; INCR_X BAR = '#'
  ; FF = '(:12:)'
  ; C = 'C'
  ; INCR_Y_BAR = '$'
  ; CR = '(:13:)'
  ; G = 'G'
  ; DOT_ = '%'
  ; CAN = '(:24:)'
  ; H = 'H'
  ; INCR_DOT = '&'
  ; EM = '(:25:)'
  ; M = 'M'
  ; VECT = '(:39:)'
  ; SUB = '(:26:)'
  ; N = 'N'
  ; CONCAT_VECTOR = '('
  ; ESC = '(:27:)'
  ; Q = 'Q'
  ; ARC = ')'
  ; FS = '(:28:)'
  ; U = 'U'
  ; CIRCLE_ = '*'
  ; GS = '(:29:)'
  ; V = 'V'
  ; RECT = '+'
  ; RS = '(:30:)'
  ; W = 'W'
  ; ZERO = '0'
  ; US = '(:31:)'
  ; X = 'X'
  ; ONE = '1'
  ; SP = '(:32:)'
  ; Y = 'Y'
  ; TWO = '2'
  ; DEL = '(:127:)'
  ; P_S = 'F'
  ; THREE = '3'
  ; P_U = 'H'
  ; P_D = 'I'
  ; FOUR = '4'
  ; M_A = 'K'
```

```
  ; M_R = 'J'
  ; FIVE = '5'
  ; D_L = 'L'
  ; D_C = 'Z'
  ; SIX = '6'
  ; D_S = '%'
  ; D_F = '#'
  ; SEVEN = '7'
  ; C_E = 'B'
  ; C_D = CR
  ; EIGHT = '8'
  ; P_C = 'O'
  ; R_C = '/'
  ; NINE = '9'
  ; P_ON = '(:1:)'
  ; P_OFF = '(:3:)'
  ; SIGNS = [ '-' , '+' ] ; CONST PAGELENGTH =
512 ; TYPE PAGE = ARRAY (. 1 .. PAGELENGTH  .)
OF CHAR ; CONST LINELENGTH = 132 ; TYPE LINE =
ARRAY (. 1  ..  LINELENGTH  .) OF CHAR ; CONST
IDLENGTH = 12 ; TYPE  IDENTIFIER  = ARRAY (. 1
.. IDLENGTH .) OF CHAR;


    "#########################################################
    "#
#"
    "# THE FOLLOWING RECORD TYPE IS THE  MEANS
BY WHICH ALL  #"
    "# DATA IS PASSED  TO  THIS PROGRAM.  THIS
PROGRAM HAS    #"
    "# BEEN TREATED  AS  AN EXTERNAL PROCEDURE
AND THIS FORM  #"
    "# OF  A  RECORD  IS  THE PARAMETER OF THE
PROGRAM CALL.    #"
    "#
#"
    "#  AN   INSTANCE   OF   THIS   RECORD  IS
INSTANTIATED IN THE     #"
    "# PROGRAM HEADING.  THE VARIABLE DECLARED
IS PARAM.    #"
    "#
#"
    "# ALL IDENTIFIERS USED WITHIN THE  RECORD
SHOULD BE      #"
    "# SELF-EXPLANATORY  AND  NEED  NO FURTHER
EXPLANATION.    #"
    "#
#"
    "#########################################################


  TYPE COORDINATE = RECORD
```

```
: ARRAY(.1..250,1..2.) OF INTEGER;
                      J_M_PLOT
: ARRAY(.1..250,1..2.) OF INTEGER;
                      COMBINATION_SELECTED
: INTEGER;
                      SCALE_DESIRED
: INTEGER;
                      NUMBER_OF_ERRORS_OBSERVED
: INTEGER
                    END;


 TYPE FILE = 1 .. 2 ; TYPE FILEKIND = (  EMPTY
, SCRATCH , ASCII , SEQCODE , CONCODE ) ; TYPE
FILEATTR
    = RECORD
        KIND : FILEKIND
      ; ADDR : INTEGER
      ; PROTECTED : BOOLEAN
      ; NOTUSED  :  ARRAY  (.  1  ..  5  .) OF
INTEGER
      END ; TYPE IODEVICE
    = ( TYPEDEVICE , DISKDEVICE , TAPEDEVICE ,
PRINTDEVICE
      , CARDDEVICE )  ;  TYPE  IOOPERATION = (
INPUT , OUTPUT , MOVE , CONTROL ) ; TYPE IOARG
= ( WRITEEOF , REWIND , UPSPACE . BACKSPACE  )
; TYPE IORESULT
    = ( COMPLETE , INTERVENTION , TRANSMISSION
, FAILURE . ENDFILE
      ,  ENDMEDIUM  ,  STARTMEDIUM  )  ;  TYPE
IOPARAM
    = RECORD
        OPERATION : IOOPERATION
      ; STATUS : IORESULT
      ; ARG : IOARG
      END ;  TYPE  TASKKIND  =  (  INPUTTASK ,
JOBTASK ,  OUTPUTTASK  )  ;  TYPE  ARGTAG  = (
NILTYPE  , . BOOLTYPE  ,  INTTYPE  ,  IDTYPE  ,
PTRTYPE ) ;  TYPE  POINTER  = @ BOOLEAN ; TYPE
PASSPTR = @ PASSLINK ; TYPE PASSLINK
    = RECORD
        OPTIONS : SET OF CHAR
      ; FILLER1  :  ARRAY  (.  1  ..  7  .) OF
INTEGER
      ; FILLER2 : BOOLEAN
      ; RESET_POINT : INTEGER
      ; FILLER3  :  ARRAY  (.  1  ..  11 .) OF
POINTER
      END ; TYPE ARGTYPE
    = RECORD
        CASE TAG : ARGTAG
        OF NILTYPE  ,  BOOLTYPE  :  (  BOOL  :
BOOLEAN )
```

```
        ; INTTYPE : ( INT : INTEGER )
        ; IDTYPE : ( ID : IDENTIFIER )
        ; PTRTYPE : ( PTR : PASSPTR )
      END ; CONST MAXARG = 10 ; TYPE ARGLIST =
ARRAY (. 1 .. MAXARG .) OF ARGTYPE ; TYPE
ARGSEQ = ( INP , OUT ) ; TYPE PROGRESULT
    = ( TERMINATED . OVERFLOW , POINTERERROR ,
RANGEERROR
      , VARIANTERROR , HEAPLIMIT , STACKLIMIT
, CODELIMIT , TIMELIMIT
      . CALLERROR ) ; PROCEDURE READ ( VAR CH
: CHAR ) ; PROCEDURE WRITE ( CH : CHAR ) ;
PROCEDURE OPEN ( F : FILE ; ID : IDENTIFIER ;
VAR FOUND : BOOLEAN ) ; PROCEDURE CLOSE ( F :
FILE ) ; PROCEDURE GET ( F : FILE ; P :
INTEGER ; VAR BLOCK : UNIV PAGE ) ; PROCEDURE
PUT ( F : FILE ; P : INTEGER ; VAR BLOCK :
UNIV PAGE ) ; FUNCTION LENGTH ( F : FILE ) :
INTEGER ; PROCEDURE MARK ( VAR TOP : INTEGER )
; PROCEDURE RELEASE ( TOP : INTEGER ) ;
PROCEDURE IDENTIFY ( HEADER : LINE ) ;
PROCEDURE ACCEPT ( VAR CH : CHAR ) ; PROCEDURE
DISPLAY ( CH : CHAR ) ; PROCEDURE READPAGE (
VAR BLOCK : UNIV PAGE ; VAR EOF : BOOLEAN ) ;
PROCEDURE WRITEPAGE ( BLOCK : UNIV PAGE ; EOF
: BOOLEAN ) ; PROCEDURE READLINE ( VAR TEXT :
UNIV LINE ) ; PROCEDURE WRITELINE ( TEXT :
UNIV LINE ) ; PROCEDURE RUN
    ( ID : IDENTIFIER
    ; VAR PARAM : ARGLIST
    ; VAR LINE : INTEGER
    ; VAR RESULT : PROGRESULT
    );
```

```
  PROGRAM PLOTFIX ( PARAM : COORDINATE );


   "###########################################################
   "#
#"
   "# PROGRAM PLOTFIX WAS ORIGINALLY  AUTHORED
BY THEODORE JOHN      #"
   "# SOCOFLOSKY, A FORMER GRADUATE STUDENT AT
KSU.  WITH THE EXCEP-#"
   "# TION OF THE PROCEDURES USER PROG, SET UP
SYSTEM, DRAW GRAPH,  #"
   "# AND OUTPUT LINE, ALL PROCEDURES USED  IN
THIS PROGRAM WERE     #"
   "# WRITTEN BY  SOCOFLOSKY.   THE PURPOSE OF
THESE PROCEDURES IS TO #"
   "# PROVIDE THE NECESSARY INTERFACE  BETWEEN
THE INTERDATA COMPUTER#"
   "# AND THE  PLOTTER  DISPLAY  DEVICE AND TO
PROVIDE THE PROGRAMMER  #"
   "#  WITH  THE  CAPABILITY  TO  OPERATE  THE
DEVICE REMOTELY.           #"
   "#
#"
   "#  THE  PROCEDURES   MENTIONED   ABOVE  AS
EXCEPTIONS WERE AUTHORED BY #"
   "# MYSELF  AND  PROVIDE  THE  MECHANISM FOR
DRAWING THE GRAPHICAL     #"
   "#     SOLUTION     ON     THE     PLOTTER.
#"
   "#
#"
   "###########################################################


 CONST    "THESE CONSTANTS REFER TO PEN  COLOR
FOR THE PLOTTER"
   BLACK       = '0';
   RED         = '1';
   GREEN       = '2';
   LIGHT_BLUE  = '3';
   DARK_BLUE   = '4';
   PURPLE      = '5';
   YELLOW      = '6';
   BLANK       = '7';


   TYPE                COMMAND_TYPE                =
(S_MOVE_CURSOR,S_X_BAR.S_Y_BAR.S_INCR_X_BAR,
                        S INCR_Y_BAR,S_DOT,S_INCR_DOT,S_VECT
                        S_CONCAT_VECTOR,S_ARC,S_CIRCLE.S_REC
                        CHARACTER);
    "$EJECT"
```

```
    VAR
      OUTPUT_COUNT          : INTEGER;
      WIDTH_CHAR            : INTEGER;
      HEIGHT_CHAR           : INTEGER;
      V_OR_H_CHAR           : INTEGER;
      Y_1                   : INTEGER;
      X_1                   : INTEGER;
      X1,X2,X3,Y1,Y2       : INTEGER;
      COMMAND               : COMMAND_TYPE;
      PEN_IS_DOWN           : BOOLEAN;
      PLOT_ON               : BOOLEAN;
      SET_BACK_ON           : BOOLEAN;
      FIRST I_X_BAR         : BOOLEAN;
      CH                    : CHAR;
      CHA                   : CHAR;
      X_OR_T            : CHAR;
      TEXT,COORD            : LINE;

  FUNCTION DCOS(RADIANS : REAL) : REAL;
    FORTRAN;

 FUNCTION DSIN(RADIANS : REAL) : REAL;
   FORTRAN;

    PROCEDURE  OUTPUT_CHAR  (  CH  :  CHAR  );
"IMAGE MODE OUTPUT"
      BEGIN "PROCEDURE OUTPUT CHARACTER"
        OUTPUT_COUNT := SUCC ( OUTPUT_COUNT );
        TEXT [ OUTPUT_COUNT ] := CH;
        IF OUTPUT_COUNT = 80
          THEN BEGIN  WRITELINE  (  TEXT  )  ;
OUTPUT_COUNT := 0 END
      END; "PROCEDURE OUTPUT CHARACTER"

    PROCEDURE FLUSH_TEXT;
      BEGIN "PROCEDURE FLUSH TEXT"
        FOR OUTPUT_COUNT :=  OUTPUT_COUNT  + 1
TO 80
        DO TEXT [ OUTPUT_COUNT ] := '(:00:)';
        WRITELINE ( TEXT );
        OUTPUT_COUNT := 0
       END; "PROCEDURE FLUSH TEXT"

    PROCEDURE OUTPUT_INT ( I : INTEGER );
      FORWARD;

    PROCEDURE OUTPUT_LINE ( STRING : LINE );
      VAR INDEX : INTEGER;
      BEGIN "TERMINATE ON NL,'$' OR 80 BUT  DO
NOT SHIP NL"
        INDEX := 1;
        OUTPUT_CHAR ( D_C );
        OUTPUT_INT ( HEIGHT_CHAR );
        OUTPUT_CHAR ( SP );
```

```
        OUTPUT_INT ( V_OR_H_CHAR );
        OUTPUT_CHAR ( SP );
        OUTPUT_INT ( WIDTH_CHAR );
        OUTPUT_CHAR ( SP );
        OUTPUT_CHAR ( C_E );
        WHILE ( STRING [ INDEX ] <> NL )
              AND ( INDEX < 81 )
              AND ( STRING [ INDEX ] <> '$' )
        DO BEGIN
          OUTPUT_CHAR ( STRING [ INDEX ] );
          INDEX := SUCC ( INDEX )
        END;
        OUTPUT_CHAR ( C_D );
        FLUSH_TEXT
      END;  "PROCEDURE OUTPUT LINE"

    PROCEDURE OUTPUT_INT;
      VAR I1000 , I100 , I10 : INTEGER;
      BEGIN "PROCEDURE OUTPUT INTEGER"
        I1000 := I DIV 1000;
        OUTPUT_CHAR ( CHR ( I1000 + 48 ) );
        I100 := ( I - I1000 * 1000 ) DIV 100;
        OUTPUT_CHAR ( CHR ( I100 + 48 ) );
        I10 := ( I - I1000 * 1000 - I100 * 100
) DIV 10;
        OUTPUT_CHAR ( CHR ( I10 + 48 ) );
        OUTPUT_CHAR ( CHR ( ( I MOD 10 ) +  48
) )
      END;  "PROCEDURE OUTPUT INTEGER"

    PROCEDURE PEN_DOWN;
      BEGIN PEN IS_DOWN := TRUE ;  OUTPUT_CHAR
( P_D ) END;

    PROCEDURE PEN_UP;
      BEGIN PEN_IS_DOWN := FALSE ; OUTPUT_CHAR
( P_U ) END;

    PROCEDURE SET_COLOR(CHA : CHAR);
      BEGIN "PROCEDURE SELECT PEN COLOR"
        OUTPUT_CHAR ( P_S );
        OUTPUT_CHAR ( CHR ( ORD ( CHA ) + 1  )
);
        OUTPUT_CHAR ( SP );
        FLUSH_TEXT
      END; "PROCEDURE SELECT PEN COLOR"

    PROCEDURE  MOVE_CURSOR  (  X_1  ,  Y_1   :
INTEGER );
      BEGIN "PROCEDURE MOVE PLOTTER CURSOR"
        OUTPUT_INT ( X_1 );
        OUTPUT_CHAR ( R_C );
        OUTPUT_INT ( Y_1 );
        OUTPUT_CHAR ( M_A )
```

```
  END;  "PROCEDURE MOVE PLOTTER CURSOR"

PROCEDURE FIX(NUMBER : INTEGER);
  VAR
    I : INTEGER;
      BEGIN "PROCEDURE FIX"
        FOR I := 1 TO NUMBER DO
          BEGIN
            "SLOW THE INTERDATA DOWN"
          END
     END; "PROCEDURE FIX"
```

```
PROCEDURE SET_UP_SYSTEM;

    "*********************************************************
    "*
*"
    "* THIS PROCEDURE IS USED TO  INITIALIZE
THE PLOTTER DIS-*"
    "*  PLAY.   THE   GRAPHICAL   FRAME  IS
INITIALIZED HERE, AS    *"
    "*  WELL  AS   THE   LEGENDS  AND  SCALE
MARKINGS.  THE ORIGIN  *"
    "*   OF    THE    GRAPH    IS   800,800.
*"
    "*
*"
    "*********************************************************


  VAR
    I : INTEGER;
      BEGIN "PROCEDURE SET UP THE SYSTEM"

      "DRAW THE X AND Y AXIS OF GRAPH"

        WIDTH_CHAR  := 36;
        HEIGHT_CHAR := 36;
        SET_COLOR(DARK_BLUE);
        PEN_UP;
        MOVE_CURSOR(800,800);
        PEN DOWN;
        MOVE_CURSOR(2800,800);
        PEN UP;
        MOVE_CURSOR(800,800);
        PEN DOWN;
        MOVE_CURSOR(800,2100);
        PEN UP;
        FLUSH_TEXT;

        "LABEL THE X AXIS OF THE GRAPH"

        FIX(300000);
        FOR I := 1 TO 10 DO
          BEGIN
            PEN_UP;
            MOVE_CURSOR((800    +    200     *
I),800);
            PEN_DOWN;
            MOVE_CURSOR((800    +    200     *
I),750);
            PEN_UP
          END;
        MOVE_CURSOR(2600,800);
        PEN DOWN;
```

```
MOVE_CURSOR(2600,750);
PEN UP;
MOVE_CURSOR(2800,800);
PEN DOWN;
MOVE_CURSOR(2800,750);
OUTPUT_COUNT := 0;
HEIGHT_CHAR := 36;
WIDTH_CHAR := 36;
OUTPUT_CHAR(P_ON);
PEN_UP;
MOVE_CURSOR(1000,650);
OUTPUT_LINE('.1$');
PEN_UP;
MOVE_CURSOR(1200,650);
OUTPUT_LINE('.2$');
PEN_UP;
MOVE_CURSOR(1400,650);
OUTPUT_LINE('.3$');
PEN UP;
MOVE_CURSOR(1600,650);
OUTPUT_LINE('.4$');
PEN_UP;
MOVE_CURSOR(1800,650);
OUTPUT_LINE('.5$');
FIX(300000);
PEN UP;
MOVE_CURSOR(2000,650);
OUTPUT_LINE('.6$');
PEN_UP;
MOVE CURSOR(2200,650);
OUTPUT_LINE('.7$');
PEN_UP;
MOVE_CURSOR(2400,650);
OUTPUT_LINE('.8$');
PEN_UP;
MOVE_CURSOR(2400,800);
PEN DOWN;
MOVE_CURSOR(2400,750);
PEN_UP;
MOVE_CURSOR(2600,650);
OUTPUT_LINE('.9$');
PEN_UP;
MOVE CURSOR(2600,800);
PEN DOWN;
MOVE_CURSOR(2600,750);
PEN_UP;
MOVE_CURSOR(2800,650);
OUTPUT_LINE('1.0$');
PEN UP;
MOVE_CURSOR(2800,800);
PEN_DOWN;
MOVE_CURSOR(2800,750);
PEN_UP;
FLUSH_TEXT;
```

```
        "LABEL THE Y AXIS OF THE GRAPH"

        FIX(300000);
        FOR I := 1 TO 11 DO
          BEGIN
            PEN_UP;
            MOVE_CURSOR(800,(800  +  100  *
I));
            PEN_DOWN;
            MOVE_CURSOR(750,(800  +  100  *
I));
            PEN_UP
          END;
        MOVE_CURSOR(800,1900);
        PEN_DOWN;
        MOVE_CURSOR(750,1900);
        PEN_UP;
        OUTPUT_COUNT := 0;
        HEIGHT CHAR := 36;
        WIDTH_CHAR := 36;
        OUTPUT_CHAR(P_ON);
        PEN UP;
        MOVE_CURSOR(650,900);
        OUTPUT_LINE('2$');
        PEN UP;
        MOVE_CURSOR(650,1000);
        OUTPUT_LINE('4$');
        PEN_UP;
        MOVE_CURSOR(650,1100);
        OUTPUT_LINE('6$');
        PEN_UP;
        MOVE_CURSOR(650,1200);
        OUTPUT_LINE('8$');
        PEN_UP;
        MOVE CURSOR(650,1300);
        OUTPUT_LINE('10$');
        FIX(300000);
        PEN_UP;
        MOVE_CURSOR(650,1400);
        OUTPUT_LINE('12$');
        PEN_UP;
        MOVE_CURSOR(650,1500);
        OUTPUT_LINE('14$');
        PEN_UP;
        MOVE_CURSOR(650,1600);
        OUTPUT_LINE('16$');
        PEN_UP;
        MOVE_CURSOR(650,1700);
        OUTPUT_LINE('18$');
        PEN_UP;
        MOVE_CURSOR(800,1800);
        PEN_DOWN;
        MOVE_CURSOR(750,1800);
        PEN_UP;
```

```
          MOVE_CURSOR(650,1800);
          OUTPUT_LINE('20$');
          PEN_UP;
          MOVE_CURSOR(800,1800);
          PEN_DOWN;
          MOVE_CURSOR(750,1800);
          PEN_UP;
          MOVE_CURSOR(620,1900);
          OUTPUT_LINE('>20$');
          PEN_UP;
          MOVE_CURSOR(800,1900);
          PEN_DOWN;
          MOVE_CURSOR(750,1900);
          FLUSH_TEXT;

          "LABEL THE Y AXIS AS MTTF"

          FIX(300000);
          PEN UP;
          MOVE CURSOR(300,1500);
          OUTPUT LINE('MTTF$');
          PEN_UP;
          MOVE_CURSOR(300,1400);
          OUTPUT_LINE(' IN$');
          PEN_UP;
          MOVE_CURSOR(300,1300);
          IF(PARAM.SCALE_DESIRED = 1)
            THEN
              OUTPUT LINE('DAYS$')
            ELSE
              IF(PARAM.SCALE DESIRED = 2)
                THEN
                  OUTPUT LINE('WEEKS$')
                ELSE
                  OUTPUT LINE('MONTHS$');
          FLUSH_TEXT;

         "LABEL THE X AXIS AS RELIABILITY"

         PEN_UP;
         MOVE_CURSOR(1600,550);
         OUTPUT_LINE('RELIABILITY$');
         PEN_UP;
         SET_COLOR(BLACK);
         MOVE_CURSOR(1200,450);
         OUTPUT_LINE('SCHICK-WOLVERTON   DATA
IS IN BLACK$');
         PEN_UP;
         SET_COLOR(GREEN);
         MOVE_CURSOR(1200,350);
         OUTPUT_LINE('JELINSKI-MORANDA   DATA
IS IN GREEN$');
         SET_COLOR(BLACK);
         PEN_UP;
```

```
      MOVE_CURSOR(0,0);
       FLUSH_TEXT
    END; "PROCEDURE SET UP THE SYSTEM"
```

```
PROCEDURE DRAW_GRAPH(WHICH_ONE : INTEGER);

    "********************************************************
    "*
*"
    "* THIS PROCEDURE  IS  USED  TO DRAW THE
GRAPHS ON THE PLOT-  *"
    "* ER DEVICE.  THE PROCEDURE ESSENTIALLY
CONTROLS THE       *"
    "* PRIMITIVE OPERATIONS  OF  THE PLOTTER
AND INCLUDES A       *"
    "*  SIMPLE   ROUTINE   TO   OBTAIN   THE
COORDINATES FROM THE PAS-  *"
    "*              SED                RECORD.
*"
    "*
*"
    "* X-AXIS COORDINATES  ARE  CONTAINED IN
COLUMN ONE OF THE   *"
    "*   PLOT   ARRAYS   AND   THE    Y-AXIS
COORDINATES ARE CONTANED IN *"
    "* COLUMN TWO OF THE PLOT ARRAYS OF  THE
RECORD TYPE.  DRAW-*"
    "* ING  THE  GRAPH  THEN  BECOMES ONLY A
MATTER OF MOVING THE  *"
    "*  PLOTTER   PEN   FROM   ONE   SET  OF
COORDINATES TO THE NEXT.    *"
    "*
*"
    "* THE CASE STATEMENT INDICATES  THROUGH
THE VALUES OF THE  *"
    "*  VARIABLE  WHICH_ONE  WHICH   MODEL'S
GRAPH TO DRAW.  A VALUE*"
    "*     OF     100     REFERENCES     THE
SCHICK-WOLVERTON MODEL, A VALUE OF*"
    "* 200  REFERENCES  THE JELINSKI-MORANDA
MODEL, AND VAUES OF *"
    "* 300  AND  400  REFERENCE  BOTH MODELS
BEING SELECTED.       *"
    "*
*"
    "********************************************************


    VAR I : INTEGER;
      BEGIN "PROCEDURE DRAW GRAPH"
        CASE WHICH_ONE OF
          100,300  : BEGIN
                       SET_COLOR(BLACK);
                       WITH PARAM DO
                         BEGIN
                           PEN_UP;
                           MOVE_CURSOR(800,800);
```

```
                              PEN_DOWN;
                              I := 1;
                              MOVE_CURSOR(S_W_PLOT(.I,1.),S_W
                              IF(NUMBER_OF_ERRORS_OBSERVED
> 1)
                                THEN
                                  FOR I := 2 TO
NUMBER_OF_ERRORS_OBSERVED DO
                                    MOVE_CURSOR(S_W_PLOT(.I,1
                              PEN_UP;
                              MOVE_CURSOR(0,0)
                            END
                          END;
            200,400   : BEGIN
                          SET_COLOR(GREEN);
                          WITH PARAM DO
                            BEGIN
                              PEN_UP;
                              MOVE_CURSOR(800,800);
                              PEN_DOWN;
                              I := 1;
                              MOVE_CURSOR(J_M_PLOT(.I,1.),J_M
                              IF(NUMBER_OF_ERRORS_OBSERVED
> 1)
                                THEN
                                  FOR I := 2 TO
NUMBER_OF_ERRORS_OBSERVED DO
                                    MOVE_CURSOR(J_M_PLOT(.I,1
                              PEN_UP;
                              MOVE_CURSOR(0,0)
                            END
                          END
          END; "CASE OF WHICH ONE"
          PEN_UP;
          SET_COLOR(BLANK);
          MOVE_CURSOR(0,0)
        END; "PROCEDURE DRAW GRAPH"
```

```
PROCEDURE USER_PROGRAM;

   "#########################################################
   "#
#"
   "# THIS PROCEDURE DETERMINES WHICH MODEL
HAS BEEN CHOSEN   #"
   "# TO PRESENT  A  GRAPH  ON THE PLOTTER.
THE KEY IS THE VAR- #"
   "# IABLE COMBINATION SELECTED, WHICH  IS
PASSED BY THE CALL-#"
   "# ING PROGRAM.  THIS VARIABLE COINCIDES
WITH THE PROGRAM  #"
   "# PROJECT'S VARIABLE NAMED DESTINATION.
#"
   "#
#"
   "#########################################################


   VAR WHICH_ONE : INTEGER;
     BEGIN "PROCEDURE USER PROGRAM"
       SET_UP_SYSTEM;
       CASE PARAM.COMBINATION_SELECTED OF
          7,10,19     : BEGIN
                          WHICH_ONE := 100;
                          DRAW_GRAPH(WHICH_ONE)
                        END;
          8,11,20     : BEGIN
                          WHICH_ONE := 200;
                          DRAW_GRAPH(WHICH_ONE)
                        END;
          9,12,21     : BEGIN
                          WHICH_ONE := 300;
                          DRAW_GRAPH(WHICH_ONE);
                          WHICH_ONE := 400;
                          DRAW_GRAPH(WHICH_ONE)
                        END
       END "CASE OF COMBINATION SELECTED"
     END; "PROCEDURE USER PROGRAM"


BEGIN "PROGRAM PLOTFIX"
      V_OR_H_CHAR := 0;
  USER_PROGRAM;
  FLUSH_TEXT "EMPTY THE OUTPUT BUFFER"
END. "PROGRAM PLOTFIX"
```

LIST OF REFERENCES

1. Goel, Armit and K. Okumoto, "Time-Dependent Error Detection Rate Model for Software Reliability and Other Performance Measures", IEEE Transactions on Reliability, Vol. R-28, No.3, August 1979, pp 206-211.

2. Hamilton, Patricia and J. Musa, "Measuring Reliability of Computation Software Centers", Proceedings of the 3d International Conference on Software Engineering, May 1978, pp 29-36.

3. Lipow, M., "On Software Reliability", IEEE Transactions on Reliability, Vol. R-28, No. 3, August 1979, pp 178-180.

4. Littlewood, B., "How to Measure Software Reliability and How Not To", IEEE Transactions on Reliability, Vol. R-28, No. 3, pp 103-110.

5. Littlewood, B., "Software Reliability Models for Modular Program Structure", IEEE Transactions on Reliability, Vol. R-28, No. 2, pp 241-247.

6. Littlewood, B., "A Reliability Model for Markov Structured Software", Proceedings of the International Conference on Reliable Software, May 1975, pp 204-207.

7. Littlewood, B., "Theories of Software Reliability: How Good Are They and How Can They Be Improved", IEEE Transactions on Software Engineering, Vol. SE-6, No. 5, September 1980, pp 489-500.

8. Miyamato, I., "Toward an Effective Software Reliability Evaluation", Proceedings of the 3d International Conference on Software Engineering, May 1978, pp 46-53.

9. Musa, John, "A Theory of Software Reliability and Its Application (Revised)", An Abstract, Bell Telephone Laboratories, Inc., pp 157-212.

10. Musa, John, "Validity of Execution Time Theory of Software Reliability", IEEE Transactions on Reliability, Vol. R-28, No. 3, August 1979, pp 181-192.

11. Putnam, Lawrence, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem", IEEE Transactions on Software Engineering, Vol. SE-4, No. 4, July 1979, pp 345-360.

12. Rubey, R., J. Dana, P. Biche, "Quantative Aspects of Software Validation", IEEE Transactions on Software Engineering, Vol. SE-4, No. 2, June 1975, pp 150-155.

13. Schick, G., R. Wolverton, "An Analysis of Competing Software Reliability Models", IEEE Transactions on Software Engineering, Vol. SE-4, No. 2, March 1978, pp 104-120.

14. Sukert, A., "Empirical Validation of Three Software Error Prediction Models", IEEE Transactions on Reliability, Vol. R-28, No. 3, August 1979, pp 199-205.

15. Sukert, A., A. Goel, "Error Modeling Applications in Software Quality Assurance", Proceedings of the Software Quality and Assurance Workshop, November 1978, pp 33-38.

16. Trvedi, A., M. Shooman, "A Many State Markov Model for the Estimation and Prediction of Computer Software Performance Parameters", Proceedings of the International Conference on Reliable Software, April 1975, pp 208-215.

17. Quantative Software Models, Data and Analysis Center for Software, March 1979, pp 3.2-3.12.

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

by

JOHNNIE OTIS RANKIN

B.S., Oklahoma State University, 1970

AN ABSTRACT OF 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

ABSTRACT

Of key interest to the Software Engineer during the program development process is the area of testing. The Software Engineer is faced with the question of determining when a sufficient level of testing has been conducted. The software reliability model has evolved as a tool to assist the Software Engineer in making this determination.

This report is an implementation of two software reliability models, the Schick-Wolverton and the Jelinski-Moranda models. The majority of this report concerns the specifics of the implementation programs, including the design factors used, the logic flow of the programs, the definitions of modular entities of the programs, and a Guide for Users of the Implementation.